Machine Learning Framework to Analyze IoT Malware Using ELF and Opcode Features

CHIN-WEI TIEN and SHANG-WEN CHEN, Institute for Information Industry
TAO BAN, National Institute of Information and Communications Technology
SY-YEN KUO, National Taiwan University

Threats to devices that are part of the Internet of Things (IoT) are on the rise. Owing to the overwhelming diversity of IoT hardware and software, as well as its variants, conventional anti-virus techniques based on the Windows paradigm cannot be applied directly to counter threats to the IoT devices. In this article, we propose a framework that can efficiently analyze IoT malware in a wide range of environments. It consists of a universal feature representation obtained by static analysis of the malware and a machine learning scheme that first detects the malware and then classifies it into a known category. The framework was evaluated by applying it to a recently developed dataset consisting of more than 6,000 IoT malware samples collected from the HoneyPot project. The results show that the proposed method can obtain near-optimal accuracy in terms of the detection and classification of malware targeting IoT devices.

CCS Concepts: • Security and privacy → Software reverse engineering; Malware and its mitigation;

Additional Key Words and Phrases: ELF analysis, IoT security, malware detection, malware classification, machine learning, opcode analysis

ACM Reference format:
Chin-Wei Tien, Shang-Wen Chen, Tao Ban, and Sy-Yen Kuo. 2020. Machine Learning Framework to Analyze IoT Malware Using ELF and Opcode Features. Digit. Threat.: Res. Pract. 1, 1, Article 5 (March 2020), 19 pages.
https://doi.org/10.1145/3378448

1 INTRODUCTION
A survey of the Internet by Cisco [1] has predicted that the number of Internet of Things (IoT) devices worldwide will increase from 18 billion in 2015 to more than 50 billion in 2020. With the continual increment in the number of connected IoT devices, threats to the devices are on the rise. Surveys by Kaspersky [2] and Kuskov et al. [3] reported that in the first half of 2018, more than 120,000 new forms of malware were discovered, three times the number in 2017 and 10 times that in 2016. Moreover, the threat posed by IoT malware is rising due to the emergence of malware targeting critical Internet infrastructures and the wide application of IoT devices in high-risk environments, such as the health and safety industries.
Due to the importance of IoT security, research and practice in this area have intensified in the past few years, but many challenges persist. Most IoT devices on the market use an outdated operating system (OS) that does not provide adequate protection against state-of-the-art anti-virus technologies compared to the Windows paradigm, where systems are frequently forced to update using the most recent OS patches and signature sets of the installed anti-virus engines. For example, weak credentials and login details enhance the vulnerability of all IoT devices being compromised through brute-force methods to obtain passwords. IoT devices comprising IoT malware such as Mirai \cite{28} and Bashlite \cite{29} have been used in some of the largest and most disruptive DDoS attacks in recent years. Moreover, to infect IoT devices across CPU architectures, IoT malware is usually compiled and launched across multiple platforms. Due to the divergence of software implementation in different platforms and the diversity of the targeted hardware, the mitigation and prevention of IoT malware is challenging.

To provide a malware diagnosis system that supports the efficient mitigation and prevention of cyberthreats in IoT devices, we propose a framework for analysis of IoT malware in this article that can characterize, detect, and classify IoT malware across multiple platforms. The key idea is to leverage machine learning to build predictive models that can detect and classify unknown malware based on the statistical features obtained from a static analysis. The first step of the analysis is a unified feature representation that can be applied to IoT binaries across multiple CPU platforms. The first group of features is obtained from a meta-analysis of the binary files and focuses on the information available in the Executable and Linkable Format (ELF) \cite{11}. The remaining features are obtained from the opcode instructions that are the outputs of reverse engineering of the binaries. In the second step, a malware detector that can differentiate IoT malware from benign binaries and a classifier that can predict the category of the detected malware are built based on the universal representation in the first step. We tested the proposed framework on an IoT malware dataset consisting of 6,251 of the most recent IoT malware specimens collected from the IoT HoneyPot project. The results show that by using state-of-the-art machine learning algorithms, the proposed framework can obtain promising results in terms of both malware detection (97% in terms of F1-measure) and malware classification (98% in terms of F1-measure).

The reminder of this article is organized as follows. Section 2 introduces previous work related to malware analysis, detection, and classification using opcode. Section 3 details the proposed method to detect and classify IoT programs, and Section 4 describes experiments to test in terms of detection and classification accuracy. Section 5 discusses the sources of false positives and false negatives, effects of extreme ratios in the test set, comparison with related work, explanation of experimental results, and the limitations of the proposed framework. Section 6 concludes the article.

2 RELATED WORK

This section is divided into two subsections: the first describes the benefits and drawbacks of static and dynamic modes of analysis, and the second describes the effectiveness of the major machine learning–based feature chosen. In addition, our contributions are described in this section.

2.1 Static and Dynamic Analyses

The major approaches to malware analysis are static \cite{6} and dynamic analyses \cite{7}. Each method has its advantages and disadvantages, and several works \cite{36,37} have discussed them in detail. The key benefit of static analysis is its speed, as it acquires the report of the given sample faster than other methods. However, the correctness of the data acquired through static methods can easily be affected by the packing technique used. With regard to dynamic analysis, malware is activated in the sandbox and system logs are captured to analyze the behavior of the sample. Therefore, the benefit of dynamic analysis is that it can provide a more complete picture of the behavior of the sample in its execution. However, dynamic analysis is more time consuming, as it requires triggering behaviors by the sample in the sandbox \cite{34} to generate the analytical report.
We chose to conduct our research using static analysis for two reasons. First, because of the rapid spread of IoT malware, an analytical report that can be generated quickly is needed to distinguish malicious samples from benign ones. Second, had we chosen dynamic analysis, we would have had to build a sandbox with the appropriate environment to execute all IoT samples. However, the instruction set architectures (ISAs) used in IoT devices are diverse, because it is difficult to build an environment flexible enough to allow for the execution of samples for all ISAs.

2.2 Machine Learning–Based Analysis

Because we acquired statistical data through static analysis for the detection and classification of malware, this information was used as features in machine learning. Many features extracted by static methods, such as byte sequence [31], strings [31, 43], DLL [31], n-gram [35], grayscale images [38], control flow graph (CFG) [39], function length frequency [40], PE header [41, 42], mnemonics [49, 50], API call [51–54], and opcode [8, 44–47], have typically been leveraged to detect and classify malware using machine learning. We select opcode as a core feature to distinguish malware from benign samples during execution. In addition, research by Bilar [8] in 2007 showed the effectiveness of opcode. Many other studies have been conducted using opcode as a feature and have yielded good accuracy in terms of the detection and classification of malware.

In 2011, Santos et al. [9] used an opcode sequence divided in accordance with n-gram as features for machine learning. They experimented with four data mining methods: decision trees, support vector machine (SVM), k-nearest neighbors, and Bayesian networks. The SVM model delivered the highest accuracy at 95%. However, they verified only the effectiveness of opcode for PE files in Windows, and that for Linux-based files or IoT malware was not verified.

In 2016, Sharma and Sahay [10] used the frequency of the occurrence of opcode as a feature, and set features with n-fold cross validation to train 13 classifiers. They also experimented with various values of n to improve accuracy and chose the top five classifiers to classify the data collected. The results showed that the best classifier was random forest, with an accuracy of 97%. Because the size of most samples in the data collected was smaller than 500 kB, observations of malware behavior were constrained. By contrast, the dataset used in this study had a wide range of sizes. Therefore, we were able to collect more detailed and varied behavioral information than Sharma and Sahay [10].

The objective of this study is to detect whether a new, unknown sample is malicious or not and classify it from among the known families of malware according its characteristics. This study makes three contributions to the literature:

- First, we collected more than 30,000 IoT malware samples and labeled 11,000 with class names from the VirusTotal [12] service. This dataset can be used as the ground truth for evaluation and other research purposes.
- Second, we collected the distributions of seven static features in the 30,000 IoT malware specimens using preferences such as the numbers and types of behaviors, the architecture targeted, shared library requirements in the system, range in size for suspicious samples, and ability to cloak characteristics through packing.
- Third, we propose the use of opcode, as divided into 12 types as the main feature for machine learning. The proposed method yielded accuracies of 97% and 98% for detection and classification, respectively.

3 PROPOSED METHOD

In this section, we introduce the proposed method in detail. First, the complete process of our framework for analysis is described, including the steps of pre-processing and processing. Second, we describe features used in the method and their detailed process of extraction. Third, we verify the effectiveness of the selected features.
3.1 IoT Malware Framework for Analysis

In keeping with the rapidly increasing number of IoT malware specimens, we combine the benefits of quick reporting due to static analysis and effectiveness of opcode features to build our system. A system overview is shown in Figure 1. In the proposed framework, the input is an executable binary in a Linux-based system, and the output is a combination of the results of detection and classification. The standard executable binary for the Linux-based system is ELF, a standard file format for executable files, object code, shared libraries, and core dumps. The detailed processing and preparation are as follows.

In pre-processing, we extract the same features, including the ELF and opcode features, from the malicious and benign samples. Features extracted from the benign samples are stored directly in the detection dataset. In contrast, labels are assigned to the malicious specimens according to their classification in keeping with reports of samples in VirusTotal [12], and information concerning them is stored in both detection and classification datasets. A total of 15 malware family names were gathered from the reports of VirusTotal: Gafgyt, Tsunami, Dofloo, ChinaZ, DdoS, DnsAmp, XorDDoS, Mirai, BitCoinMiner, Elknot, Ganiw, Ramgo, Flood, Flooder, CornelGEN, and AESDDoS. Of the two datasets mentioned previously, the detection dataset contained both benign and malicious samples, and the other was an independent dataset composed of malicious samples. The dataset containing features of malware was used by the classification model, and that containing features of both malicious and benign samples was used by the detection model.

Following pre-processing, if an unknown new sample enters our system, the detection model determines whether it is malicious. If it is, it is entered into the classification model to identify the malware family to which it belongs from our dataset, where this model reports both the result of detection and classification. If the sample is benign, the output of the system shows only the result of detection.

We now discuss the composition of our datasets and features extracted for use in machine learning. We detail how these features can be used to identify and classify IoT malware.

3.2 ELF Features

We extract seven ELF features from the samples: ISA, size, external library, packer, number of functions, networking, and other abilities. These features are described in Table 1. The third feature reflects whether the Linux
Table 1. Description of ELF Features

| ELF Features  | Description                                                                 |
|---------------|-----------------------------------------------------------------------------|
| ISA           | Abstract model of a computer where the sample can be executed               |
| Size          | Size of the sample; unit KB                                                 |
| External library | Related library that can be called during execution                        |
| Packer        | Checking whether the sample is packed                                       |
| Number of functions | Number of functions in the sample disassembled by tools                  |
| Networking    | Possibility of connecting the network for sample execution                 |
| Other abilities | Potential behavior judged by known YARA rules                           |

command `ldd` needs related common libraries. If there is no common library, the value is 1, and if there is, the value is zero. The fourth feature shows whether the sample is packed according to YARA rules [13]. Radare2 [26] is a portable reversing framework that can disassemble several architectures. We use the command `afl` of Radare2 to disassemble the function lists and obtain the number of functions in the sample represented by the fifth feature. For networking, we cannot confirm the connection with the Internet by executing the sample directly. Instead, strings and function names extracted from the samples are used to gauge this capability. The function names are acquired from the function lists that are extracted by Radare2, and strings are extracted by a program named `strings` in the elfutils project [27], which is a collection of utilities and libraries to read, create, and modify ELF binary files. If IP addresses or domain names are found in the strings and protocol names are found in function names at the same time, this implies that the binary has implemented the network function and the feature value is set to 1, and is otherwise set to zero. The seventh feature assesses whether samples may exhibit other behaviors, such as anti-VM detection and using known exploit-kits attack. We assess behaviors using the YARA rules [13].

3.3 Opcode Features

The malicious samples in our dataset were designed for eight ISAs: x86, x64, MIPS, ARM, SPARC, PowerPC, Renesas SH, and Motorola m68k. Each ISA has specific opcodes, but some with different spellings have similar functions. Therefore, we divide opcodes with different ISAs into 12 types according to their description files [14–21]: logic, control and status, memory, stack, procedure, prefixed, system input and output, arithmetic, system, branch, execution timing, and the others. Example of opcodes for different ISAs and types are shown in Table 2. The same process was applied to the benign samples.

3.4 Effectiveness of Features

Next, we detail the statistics of features of ELF and opcode to determine whether they can distinguish between malware and benign samples, and can reflect trends in the distribution of the samples.

We used 30,146 IoT malware specimens collected by the IoT HoneyPot project of Yokohama National University [4] and CZ.NIC [5] from 2016 to 2018, where 25,000 specimens were from the former and 5,146 from the latter. Moreover, the IoT HoneyPot project of Yokohama National University and CZ.NIC utilized that their machine of IoT HoneyPot simulated a real IoT device instead of installing third-party software into an IoT device. Thus, we did not have to consider how to deploy software into a real IoT device. We also collected 2,157 benign samples from the Linux-based operation system Ubuntu 16.04 LTS Client, which was composed of three architectures: x86, x64, and ARM. These benign samples were collected from the system locations “/bin,” “/sbin/” and “/usr/bin,” and were used to save the executable files of the Linux-based system. Although we collected benign samples from Ubuntu instead of IoT devices, there were still enough common features that we could extract from them. This is because the Ubuntu system and the OS of the IoT device are all Linux-like systems.
Table 2. Example of Opcode Types

| Type | Logic | Control and Status | Memory | Stack | Procedure | Prefixed | System I/O | Arithmetic | System | Branch | Execution Time | Others |
|------|-------|---------------------|--------|-------|-----------|----------|------------|------------|--------|--------|----------------|--------|
| ISA  | Type1 | Type2 | Type3 | Type4 | Type5 | Type6 | Type7 | Type8 | Type9 | Type10 | Type11 | Type12 |
| x86  | OR    | CMP    | MOV   | PUSH  | CALL   | ES     | IN     | ADD      | HLT    | JMP    | WAIT           | CBW    |
| x64  | XOR   | TEST   | LEA   | POP   | LEAVE  | DS     | OUT    | SUB      | ARFL   | RETN   | FWAIT          | CDQ    |
| MIPS | nor   | slti   | lw    | break |        |        | multi  |          |        |        |                 |        |
| ARM  | BIC   | CMN    | LDC   | SRS   | HVC    | ASR    | SYS    | BL       | WFE    | DSB    |                |        |
| SPARC| xor   | PCMP   | restore | CALL |        |        |        |          |        |        |                 |        |
| PowerPC | xori | debi | lwisi | svc |        | abs   | ti     | bclr     |        |        |                 |        |
| Renesas_SH | NOT | TAS | STS |         |        |        | MAC    |          |        |        |                 | BSRF   |
| Motorola m68k | EOR | RTST | USP | PEA |        |        |        |          |        |        |                 |        |

Fig. 2. Distribution of 30,146 malware samples across architectures.

3.4.1 Effectiveness of ELF Features. Figure 2 shows the distribution of malware samples for various ISAs. According to the figure, the three most popular ISAs are x86, MIPS, and ARM. The architecture of x86 is backward compatibility, whereas ARM has benefits in terms of saving energy. Moreover, the MIPS architecture was widely used in network gateways and routers when first developed. As the malware binaries were collected from various distributions of ISAs, we included benign samples from different ISAs in the dataset. The samples were collected from the Ubuntu OS from the ISAs of x86, x64, and ARM. Owing to technical difficulties in the process, the dataset contained a limited number of successful samples from ARM. More benignware will be added to the dataset in future work.

We surveyed the distribution of sizes of the samples, both malicious and benign, and the results show marked differences within the ranges from 1,000 to 1,200 kB and 1,800 to 2,000 kB. Note also that regardless of the malicious or benign nature of the samples, the range of sizes under 200 kB was the most populated. We consider this range more closely, and the results are shown in Figure 3. The 0 to 200 kB range was divided into 10 smaller
ranges at intervals of 20 kB. Moreover, percentages for each range of sizes in Figure 3 were the ratio of the number of malware/benign samples in a given range to the total number of malware/benign samples. Figure 3 shows a clear delineation of the five ranges. Therefore, size can be used to distinguish between malware and benign samples. Although more than half of the sample in the range under 40 kB were benign, there were only 11% malware samples in this range. This fact could be quite different from generally common thinking that malware is small but has complex ability. But our collecting malware samples from the source has VirusTotal reports to make sure that they are malware exactly.

The ratio of the number of samples with dependent external libraries to the total number of samples is shown in Figure 4. It reveals that malware does not often tend to call external libraries. According to the message of the section header extracted by the tools, dynamic relational headers were mostly absent from the information extracted on malicious samples. Therefore, most of the samples were assumed to be statically connected. The distribution of the number of functions is shown in Figure 5. Remarkable differences between malware and benign samples were observed in half of the ranges. In addition, the cumulative percentage of these ranges was
approximately 70%. Therefore, the number of functions can be used to separate benign from malicious samples. Figure 6 shows the ratio of the number of samples that may try to connect to the network to obtain malicious data or fetch commands from the attacker to the total number of samples. It indicates that the malware tends to connect to the network regardless of its purpose because the functions of IoT are mostly related to connecting networks. However, we cannot activate samples directly, because we use the static analysis method. Instead, we combine the strings and protocol names that are taken out by tools of Radare2 and elfutils to judge that whether sample tries to connect network when it executes in system. Therefore, if tools cannot take strings or protocol names out successfully, the decision and the final ratio of samples will be affected.

Digital Threats: Research and Practice, Vol. 1, No. 1, Article 5. Publication date: March 2020.
3.4.2 Effectiveness of Opcode Features. To demonstrate the effectiveness of detection and classification using opcode features, we used the examples shown in Figures 7 and 8. Figure 7 shows the ratio of the numbers of opcode in each type to the numbers of total opcode in the sample. Marked differences were observed for several types of malicious and benign samples, showing that the proposed method can distinguish between malware and benign samples. Because the categories of opcode are varied and its functions are complex, we manually identified 11 classes that had a significant influence on opcode, and the remaining opcodes were classified as "others." Therefore, the amounts of opcodes belonging to the "others" class are more than the preceding 11 classes. Figure 8 similarly shows differences among malware families. Therefore, this proposed method can also distinguish families of malware.
In this section, we describe the details of the experiments involving detection and classification using different methods and conditions in our system. Furthermore, we discuss the process of data labeling and the need for it, the environment for the experiment, detailed configurations of the model, change in the composition of the test set, and the results of the experiments.

4.1 Data Labeling
The original dataset was composed of more than 30,000 malwares and 2,000 benign samples. To build a system that can detect and classify IoT malware in an automated manner, we leverage a supervised learning method to explore the discriminating information in the numerical features defined in the previous section. For learning and evaluation, the malware samples in the dataset were labeled with their malware family names. The labeling process consisted of three stages. First, we downloaded reports from VirusTotal for each sample to label it with the name of the malware family to which it belonged. If we could not find a report for a given sample, it was removed from the dataset. Second, there were more than 50 vendors in the report, and the naming convention of each was different. To determine the family name of a given sample, we selected the results for the three most accurate vendors according to a survey [22] by AV-Test. This sample was labeled with its family name if there was a consensus among the three vendors. Otherwise, it was removed from the dataset. Finally, we kept only malware samples belonging to the same ISAs of the benign samples to avoid learning bias created by samples from different ISAs. The process is shown in Figure 9. At the end of the labeling process, 6,251 labeled malware samples were also in the dataset. A total of 2,157 benignware samples were also in the dataset. Because of a non-disclosure agreement, we cannot provide the malicious samples directly. However, metadata of the samples for the interested reader is available.\footnote{https://drive.google.com/open?id=1NXNsFYa2Qz0cZEy3u3bK_kx4SVUrykAz.}

4.2 Experimental Setup
In this section, we describe the detailed setup of the experiments. First, the necessary information about our machine is provided, including its hardware and software. Second, the detailed configurations of our models of machine learning are described. Third, the changing ratios of the malware and the corresponding benignware in the test set are described.

Our machine consisted of four processors and had 4 GB of memory. The OS used was Ubuntu 16.04 LTS Client 64-bit, and the programming language was Python 3.5.2. The toolkits used were scikit-learn and Anaconda [23].
and the frameworks used for machine learning were Keras [24] and TensorFlow [25]. The methods used for machine learning were ANN, CNN, and SVM.

Our model had five layers. We used a dropout layer that can reduce parts of output data of the previous layer randomly to avoid overfitting. The probability of the dropout layer was set to 60% between each layer. Moreover, we select the exponential linear unit (ELU) as the activate function for the first through fourth layers and softmax as the activate function for the last layer to generate results of prediction. The optimizer used was Adam.

If we had divided the dataset for detection into training and test sets in proportion of the composition of the original dataset, the ratio of malware to benign samples in the test set would have been higher than in practical scenarios. Therefore, under a constraint whereby the total numbers of malware and benign samples remained constant, we experimented with various ratios to identify the ones that yield the highest F1-scores. Our model was also the most stable at this ratio.

4.3 Experimental Results

To examine the effectiveness of the opcode features, we carried out learning both with and without them. In each experiment, we used three machine learning methods and two scenarios. The first experiment used different numbers of samples for the training set and the remainder as a test set. The composition of samples in each set was determined by a random seed. By comparing the values obtained with and without adding opcode features, it was clear that the F1-score increased when opcode features were used. Therefore, we chose the CNN, which had yielded the highest F1-score, for machine learning in the system.

The second experiment was a repetition of the first with a 10-fold cross validation to use all data in the dataset. The average values are shown in Figure 10. The chart for various proportions in the figure indicates that the highest F1-score was obtained for a ratio of 10% of malware to 90% of benign samples, although this trend did not match the results of the first experiment. Because of the clear difference in the ratio of malware to benignware in our dataset, we use the confusion matrix of our experiment to prove the credibility of the proposed method. However, 10 different confusion matrices were obtained in the 10-fold cross validation. We used the average of each element in all matrices. Table 3 shows the average score of the 10-fold confusion matrix of the CNN at a ratio of 10% malware and 90% benign samples.
In Table 4, the results of classification for all families show the effectiveness of the opcode features. The CNN delivered the highest accuracy and F1-score. Our classification experiments were directed not only toward all families but also toward each family. These results of classification for each family are shown in Figure 11. The 10 most accurate scores for each family of methods are shown. Although the accuracy of each family varied, they were all above 97%. This indicates that our method is effective for each of the families of malware.

5 DISCUSSION
In this section, several issues are discussed. They include false positives and false negatives in the classification of malware families, influence of the tuning ratio of malware to benignware in the test set, comparison with related work, reasons for the effectiveness of the classification of malware families, ISA neutrality for detection, and the limitations of our method.
5.1 False Positives and False Negatives

In classification experiments for each family, the effectiveness of our proposed method was shown. However, Figure 11 shows that some false positives and false negatives persisted.

The reason for the preceding might have been the similarity in the ratios of opcodes among malware families. Figure 12 shows four samples from different malware families. The leftmost frame represents the opcode parameters of the samples, the central frame shows the labeled family name, and the rightmost frame shows the result of prediction by the model. It is apparent that parameters in the leftmost frame for each sample are similar, but the malware families to which they belong are all different. From the perspective of the family DnsAmp, the samples of families ChinaZ and Xorddos were false positives, and its own sample was a false negative.

One solution to the preceding is to find more effective features to add as parameters to distinguish between similar samples. For example, we can divide the ranges of parameters into finer intervals to separate similar samples.

5.2 Extreme Ratio of the Test Set

In the experiment where the ratio in the test set was changed, we tested five different ratios between malicious and benign samples. These ratios are shown in Figure 10, but not contain the ratio of 1% malware samples and 99% benign samples. Finally, we selected the set consisting of 10% malware samples and 90% benign samples. However, the $F_1$-score can be affected by such an extreme ratio. To verify that our method is not easily influenced by such ratios, we used a test set consisting of 1% malware and 99% benign samples. The same validation experiment was performed as before (i.e., the same process was repeated with 10-fold cross validation), and the results are shown in Figure 10.

As shown in Figure 10, the $F_1$-score was influenced by the extreme ratio, but the shift in its value was smaller than 1%. This proves that our method can withstand extreme ratios in the test set. In other words, the performance of our method can be adopted into many situations.

5.3 Comparison with Related Work

We compare our method to previously proposed methods in the literature that use machine learning to detect malware. The results are shown in Table 5 in terms of five perspectives of research: accuracy, $F_1$-score, algorithm, the number of malware samples, and the main features used.

Azmoudeh et al. [30] used opcodes as the main feature of machine learning. They selected the CNN for machine learning, and the accuracy and $F_1$-score recorded by their method were superior to ours. However, the number of malware samples they used was considerably smaller than in our experiment, which suggests that their results did not comprehensively reflect scenarios involving malware. In addition, our samples are collected by the IoT HoneyPot distributed in the real network. In other words, our dataset can reflect scenarios involving malware more comprehensively and the real trend of IoT malware more accurately.
Table 5. Comparison with the Past Research

| Study            | Accuracy | F1-Score | Algorithm  | Amount of Malware | Main Feature |
|------------------|----------|----------|------------|-------------------|--------------|
| Azmoudeh et al. [30] | 99.68%   | 98.48%   | CNN        | 128               | Opcodes      |
| Santos et al. [9]  | 95.91%   | 81.75%   | SVM        | 13,189            | Opcodes      |
| Schultz et al. [31] | 97.11%   | 97.99%   | Naive Bayes| 3,265             | Byte sequence|
| Tian et al. [32]  | 97.3%    | 96.9%    | Random forest| 1,368          | API call sequences |
| This work         | 99%      | 97%      | CNN        | 6,251             | Opcodes      |

Santos et al. [9] also used opcodes as the main feature for machine learning but used the $n$-gram method to obtain the frequency of the opcode sequence. They tried different algorithms and different values of $n$, and obtained the highest accuracy when the SVM algorithm was used for machine learning and $n$ was 2. Based on the concept of SVM and their database, which contained almost the same number of malware and benign samples, the effects of SVM should have been more prominent than other machine learning algorithms. However, even with an accuracy of 95%, the F1-score of their method was only 81.75%. This indicates that it yielded more false positives and false negatives than our method.

Schultz et al. [31] converted a binary file into byte sequence by using the command `hexdump` in Linux and placed the sequence and other features, like strings and system information, using a naive Bayes algorithm. This algorithm assumes that the input features are independent. Because of this, the accuracy and F1-score obtained by their method were high. However, the length of the byte sequence used as input must be a fixed number, and one of their assumptions was that the malware and benign samples exhibited similar byte code sequences. In contrast to their assumptions and the limitation of their method, our method does not have any such limit in terms of features. Moreover, its results were similar to or better than the method of Schultz et al. [31] and even used more samples.

Tian et al. [32] used the dynamic API call sequences as the main feature and selected random forest as their algorithm. They used a tool to acquire API call sequences of the sample in the sandbox, then put them into a random forest classifier to detect whether the sample was malicious or not and classify it into the malware family to which it belonged. However, their method adopts dynamic analysis, and their average analysis time for the sample will be more than ours. In addition, our results are the same or better than theirs even if our sample amount is more than theirs.

5.4 Results

We investigate reasons for the high rate of detection of malware of our method. After trying various combinations of features, we used sample size and the number of functions in the program as being crucial to its performance. We show the features in the two-dimensional graph in Figure 13. To minimize the influence of an unbalanced number of samples of malware families, we used the number of samples of malware families to plot the graph. For ease of comprehension, we show only the three most populated malware families; distribution of the samples is shown in Figure 13. The blue points represent samples of the Gafgyt malware family, which is also called Bashlite; the green points represent samples of Tsunami; and the red points represent samples of the Dofloo malware family.

It is evident from the figure that the red points were far from the points in the other colors. This means that the size and the number of functions of the samples belonging to Dofloo were considerably greater than those of the others. This can be explained by its structure of its agent handler and the characteristic of the SendInfo thread that derives the computation power of the infected host device, thus enabling the CNC Server to tune the intensity of DDoS jobs that each bot should perform. Therefore, the control functions of the SendInfo thread are more complex than those of traditional threads. In addition, the corresponding size increased with the number of functions.
In Figure 13, the other two malware families are concentrated in the bottom left and are separated into two clusters. The figure also shows two clusters of samples separated by the line indicating 500 functions of the samples. The cluster above the line indicating 500 functions of samples is composed mostly the green points (i.e., the Tsunami malware family). In addition, the rest of the green points are mixed with the blue points in the other cluster. From the perspective of the number of functions of samples, samples of Tsunami were separated into two clusters. By contrast, from the perspective of the size of the samples, they were distributed in a continuous, wide range. This distribution can be explained by the origin of this family. Tsunami is an offspring of Linux.Hydra and the result of the fusion of DDoS-Kaiten Trojan and Chuck Norris. It exhibits various behaviors in the context of attacks, such as the dictionary attacks and XMAS attacks. According to different combinations of attacks, the size of the sample was distributed over a continuous and wide range. Moreover, the number of functions was determined by the complexity of different combinations of attacks. Thus, more complex attacks might have occurred, and the differences between the clusters were determined by whether these attacks were contained.

The cluster below the line indicating 500 functions of samples is primarily composed of blue points (i.e., the Gafgyt malware family). Figure 13 shows that the distribution of samples of Gafgyt is concentrated in the area along the x-axis at values lower than 300 and along the y-axis at values lower than 400. This means that the size of most samples in this family was smaller than 300 kB and the number of functions was smaller than 400. There are two possible explanations for this situation. First, the communication protocol of Gafgyt is a lightweight version of Internet relay chat (IRC) [33]. This means that the requisite functions and corresponding sizes of samples decreased prominently. Second, even if the samples of Gafgyt can infect multiple ISAs, it can be used to launch basic DDoS attacks, such as SYN and UDP attacks. Thus, it can use the optimized code of these attacks to reduce the size of the samples and number of functions.
5.5 ISA Neutrality for Detection and Classification

Because our dataset featured various ISAs, we checked whether the results were influenced by a specific ISA—in other words, whether our framework is influenced by the ISA. Our experiments were divided according to two aspects: detection and classification. Both experiments involved choosing samples from a specific ISA as the training set and those from another as the testing set. Then model was then trained using samples from the former and tested on the latter.

We selected samples from the ISA x86 as the training set and those from the other ISAs in our dataset by turns as the testing set. Table 6 shows the results. Although the values of accuracy, precision, recall, and F1-score were satisfactory, the value of val_loss was too high to mirror practical situations. Therefore, we checked the confusion matrix and found that the results were mostly for the Gafgyt family. There are two possible reasons for this. One is the unbalanced number of samples of each malware family. There were more samples of Gafgyt than the others, because of which the training model tends to predict samples as belonging to the Gafgyt family. The other reason is that the samples of all malware families did not exist in all of the collected samples of ISAs at the same time. Thus, when testing samples, the representatives of whose families were absent from the training set, the model tended to predict that they belonged to the Gafgyt family. Thus, we limit our experimental samples to samples of common families between two ISAs. We thus limited the experimental samples to those belonging to families that were common between ISAs and ran the experiment again. The composition of the training set was the same as before. We show results of the testing set composed of samples of x64. Table 7 shows that the results improved. This shows that if we had collected more samples for each ISA, we could have achieved the ISA neutrality in classification.

We then performed an experiment on detection. The training set consisted of samples of x86, and samples of x64 were used as the testing set. As Table 8 shows, the results were not considerably worse than those of the original detection. The F1-score was nearly 83%. We can infer two results from this. First, the decreased value of the F1-score proves that the ISA has a clear influence on the results of detection. Second, the model trained by samples of the same ISA can be used to predict samples of other ISAs. This result is different from that of
the preceding classification experiment. Thus, difference obtained reason because of the smaller difference in number between malicious and benign samples. Therefore, the proposed method is neutrality to the ISA of the samples.

As such, we need to collect a greater variety of malware samples of families for each ISA to enable the proposed method for classification to samples in each ISA.

5.6 External Validity

To validate the effectiveness of the framework, we collected new samples from third-party sources. We collected 24 samples belonging to malware families already in our dataset \([55]\) as a new testing set. They consisted of 23 samples of Mirai and one of Tsunami.

The results of validation were divided into those of detection and classification, and they are shown in Table 9. Our framework clearly performed well on detection according to the confusion matrix. However, its results for classification were much worse than average. We believe that this is because our model contained bias owing to the normal distribution of these families. We ran the experiment again to confirm this assumption by placing half of the testing samples into the training set. If the characteristics of testing samples can be extracted into a new model, the result of classification should be improved obviously. As the confusion matrix in Table 9 shows, the result of classification of the retried experiment has improved clearly. The result proves that our assumption should be true. Thus, if we collect more different kinds of samples of a family, our model can learn more characteristics and raise the classification rate.

5.7 Limitations

One limitation of our proposed system is the anti-disassembly technique used. In our framework, the features of new samples are extracted using various tools and commands. Opcode—the main feature in the system—is acquired from the assembly file extracted by IDA Pro. If a sample is processed by an anti-disassembly technique, its opcode cannot be acquired, which affects the \(F_1\)-score for detection and classification.

Moreover, there is a kind of malware called *droppers*. Its purpose is to deliver enclosed payloads to the target computers. In other words, a dropper is just a means to an end rather than an end itself, and it is used in the early stages of malware attacks. The main malicious behaviors are activated by the delivered payload, which is almost independent of the dropper itself. Moreover, the behavior of the activated payload can be detected only through dynamic analysis. Thus, if our framework cannot detect the writing of the payload into the file system, there is no other way to detect it.
Furthermore, our system has 19 features: 7 ELF features and 12 opcode features. Although we can confirm the effectiveness of opcode, each feature has a different influence on performance. In future work, we will search for the most influential features when identifying IoT malware.

6 CONCLUSION

In this article, we gather the distributions of seven features of more than 30,000 malware specimens. These distributions show malware preferences including the possible number and types of behaviors, key targeted architecture, requirement for shared libraries, size range for suspicious samples, and cloaking ability for characteristics through packing. Furthermore, we propose an effective method using opcode and ELF characteristics as features for machine learning to detect and classify unknown IoT samples. We then implement a new use for opcode, where we divide opcodes into 12 different types according to their functions. In closing, our method proves to be effective and yields an F1-score of 97% and an accuracy of 98% for detection and classification.

REFERENCES

[1] D. Evans. 2011. The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. Retrieved January 31, 2020 from https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.
[2] Kaspersky. 2018. New IoT-Malware Grew Three-Fold in H1 2018. Retrieved January 31, 2020 from https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018.
[3] V. Kuskov, M. Kuzin, Y. Shmelev, D. Makrushin, and I. Grachev. 2017. Honeypots and the Internet of Things. Retrieved January 31, 2020 from https://securelist.com/honeypots-and-the-Internet-of-things/78751/.
[4] Yokohama National University. (n.d.). Home Page. Retrieved January 31, 2020 from http://www.ynu.ac.jp/.
[5] CZ.NIC. (n.d.). Home Page. Retrieved January 31, 2020 from https://www.nic.cz/.
[6] D. Wagner. 2000. Static Analysis and Computer Security: New Techniques for Software Assurance. Ph.D. Dissertation. University of California at Berkeley.
[7] U. Bayer, A. Moser, C. Krügel, and E. Kirda. 2006. Dynamic analysis of malicious code. Journal in Computer Virology 2, 1 (2006), 67–77.
[8] D. Bilan. 2007. Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics 1, 2 (2007), 156–168.
[9] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences 231 (2013), 64–82.
[10] A. Sharma and S. K. Sahay. 2016. An effective approach for classification of advanced malware with high accuracy. International Journal of Security and Its Applications 10, 4 (2016), 249–266.
[11] Wikipedia. 2020. Executable and Linkable Format. Retrieved January 31, 2020 from https://en.wikipedia.org/wiki/Executable_and_Linkable_Format.
[12] VirusTotal. (n.d.). Home Page. Retrieved January 31, 2020 from https://www.virustotal.com.
[13] GitHub. 2020. Yara-Rules/rules: Repository of Yara Rules. Retrieved January 31, 2020 from https://github.com/Yara-Rules/rules.
[14] coder32 edition | X86 Opcode and Instruction Reference 1.12. Retrieved January 31, 2020 from http://ref.x86asm.net/coder32.html.
[15] geek64 edition | X86 Opcode and Instruction Reference 1.12. Retrieved January 31, 2020 from http://ref.x86asm.net/geek64.html.
[16] ARM. 2001. ARM Information Center. Retrieved January 31, 2020 from http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0068b/CIHE2HF.html.
[17] C. Price. 1995. MIPS IV Instruction Set. Retrieved January 31, 2020 from https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf.
[18] Sun Microsystems. 1998. SPARC Assembly Language Reference Manual. Retrieved January 31, 2020 from https://docs.oracle.com/cd/E19620-01/805-4694/805-4694.pdf.
[19] IBM. 1994. The PowerPC Architecture: A Specification for a New Family of RISC Processors. Retrieved January 31, 2020 from http://bitsavers.trailing-edge.com/pdf/ibm/powerpc/SP28-5124-01_The_PowerPC_Architecture_May94.pdf.
[20] Motorola. (n.d.). Motorola M68000 Family Programmer’s Reference Manual.
[21] Hitachi. 1996. SuperH RISC Engine SH-1/SH-2 Programming Manual. Retrieved January 31, 2020 from https://antime.kapsi.fi/segas/files/h12p.pdf.
[22] AV-Test. 2015. Linux: 16 Security Packages Against Windows and Linux Malware Put to the Test. Retrieved January 31, 2020 from https://www.av-test.org/en/news/linux-16-security-packages-against-windows-and-linux-malware-put-to-the-test/.
[23] Anaconda. (n.d.). Anaconda Distribution (Download). Retrieved January 31, 2020 from https://www.anaconda.com/download/.
[24] Keras Documentation. (n.d.). Home Page. Retrieved January 31, 2020 from https://keras.io/.
[25] TensorFlow. (n.d.). Home Page. Retrieved January 31, 2020 from https://www.tensorflow.org/.
[26] Radare. (n.d.). Home Page. Retrieved January 31, 2020 from https://rada.re/.
Machine Learning Framework to Analyze IoT Malware Using ELF and Opcode Features

[55] Stratosphere Lab. (n.d.). Malware on IoT Dataset. Retrieved January 31, 2020 from https://www.stratosphereips.org/datasets-iot.

Received June 2019; revised December 2019; accepted January 2020

Digital Threats: Research and Practice, Vol. 1, No. 1, Article 5. Publication date: March 2020.

[27] Elfutils. (n.d.). Home Page. Retrieved January 31, 2020 from https://sourceware.org/elfutils/.

[28] Wikipedia. 2019. Mirai (Malware). Retrieved January 31, 2020 from https://en.wikipedia.org/wiki/Mirai_(malware).

[29] Wikipedia. (n.d.). BASHLITE. Retrieved January 31, 2020 from https://en.wikipedia.org/wiki/BASHLITE.

[30] A. Azmoudah, A. Dehghantanha, and K. K. R. Choo. 2017. Robust malware detection for Internet of (Battlefield) Things devices using deep eigenspace learning. IEEE Transactions on Sustainable Computing 4, 1 (2017), 88–95.

[31] M. G. Schultz, E. Eskin, F. Zadok, and S. Stoïfo. 2001. Data mining methods for detection of new malicious executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy. 38–49.

[32] R. Tian, L. Batten, and S. Versteeg. 2010. Differentiating malware from cleanware using behavioural analysis. In Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE’10). 23–30.

[33] Wikipedia. 2020. Internet Relay Chat. Retrieved January 31, 2020 from https://en.wikipedia.org/wiki/Internet_Relay_Chat.

[34] Sandbox. 2020. Sandbox (Computer Security). Retrieved January 31, 2020 from https://en.wikipedia.org/wiki/Sandbox_(computer_security).

[35] J. Z. Kolter and M. A. Maloof. 2004. Learning to detect malicious executables in the wild. In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, 470–478.

[36] E. Gandotra, D. Bansal, and S. Sofat. 2014. Malware analysis and classification: A survey. Journal of Information Security 5, 2(2014), 56.

[37] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar. 2017. A survey on malware detection using data mining techniques. ACM Computing Surveys 50, 3 (2017), 41.

[38] N. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath. 2011. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security. Article 4.

[39] D. Kong and G. Yan. 2013. Discriminant malware distance learning on structural information for automated malware classification. In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. 347–348.

[40] R. Tian, L. Batten, and S. Versteeg. 2008. Function length as a tool for malware classification. In Proceedings of the 3rd International Conference on Malicious and Unwanted Software. 57–64.

[41] T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh. 2009. Detecting unknown malicious executables using portable executable headers. In Proceedings of the 5th International Joint Conference on INC, IMS, and IDC (NCM’09). 278–284.

[42] A. Walenstein, D. J. Hefner, and J. Wichers. 2010. Header information in malware families and impact on automated classifiers. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software (MALWARE’10). 1522.

[43] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao. 2008. SBMDS: An interpretable string based malware detection system using SVM ensemble with bagging. Journal of Computer Virology and Hacking Techniques 5, 4 (2008), 283–293.

[44] M. Alazab, M. A. Kadiri, S. Venkataraman, and A. Al-Nemrat. 2012. Malicious code detection using penalized splines on OPCODE frequency. In Proceedings of the 2012 3rd Cybercrime and Trustworthy Computing Workshop (CTC’12). 38–47.

[45] X. Hu, S. Bhatkar, K. Griffin, and K. G. Shin. 2013. MutantX-S: Scalable malware clustering based on static features. In Proceedings of the USENIX Annual Technical Conference (USENIX ATC’13). 187–198.

[46] R. Moskovitch, C. Feher, N. Tschar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici. 2008. Unknown malcode detection using OPCODE representation. In Intelligence and Security Informatics. Lecture Notes in Computer Science, Vol. 5376. Springer, 204–215.

[47] N. Runwal, R. M. Low, and M. Stamp. 2012. Opcode graph similarity and metamorphic detection. Journal of Computer Virology and Hacking Techniques 8, 1–2 (2012), 37–52.

[48] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. 2006. Polymorphic worm detection using structural information of executables. In Recent Advances in Intrusion Detection. Lecture Notes in Computer Science, Vol. 3858. Springer, 207–226.

[49] M. R. Chouchane and A. Lakhotia. 2006. Using engine signature to detect metamorphic malware. In Proceedings of the 4th ACM Workshop on Recurring Malcode (WORM’06). ACM, New York, NY, 73–78.

[50] X. Hu, T.-C. Chiuhe, and K. G. Shin. 2009. Large-scale malware indexing using function-call graphs. In Proceedings of the 16th ACM Conference on Computer and Communications Security. 611–620.

[51] M. Christodorescu, S. Jha, and C. Kruegel. 2008. Mining specifications of malicious behavior. In Proceedings of the 1st India Software Engineering Conference (ISEC’08). ACM, New York, NY, 314–318.

[52] A. Pfeffer, C. Cail, J. Chamberlain, L. Kellogg, J. Oueltette, T. Patten, G. Zacharias et al. 2012. Malware analysis and attribution using genetic information. In Proceedings of the 7th IEEE International Conference on Malicious and Unwanted Software (MALWARE’12). IEEE, Los Alamitos, CA, 39–45.

[53] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. 2007. Automated classification and analysis of Internet malware. In Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection (RAID’07). 178–197.

[54] P. Trinius, C. Willems, T. Holz, and K. Rieck. 2009. A Malware Instruction Set for Behavior-Based Analysis. Technical Report. University of Mannheim.