Robust Malware Detection for Internet of (Battlefield) Things Devices Using Deep Eigenspace Learning

Amin Azmoodeh, Ali Dehghantanha, *Senior Member, IEEE*, and
Kim-Kwang Raymond Choo, *Senior Member, IEEE*

Abstract—Internet of Things (IoT) in military settings generally consists of a diverse range of Internet-connected devices and nodes (e.g., medical devices and wearable combat uniforms). These IoT devices and nodes are a valuable target for cyber criminals, particularly state-sponsored or nation state actors. A common attack vector is the use of malware. In this paper, we present a deep learning based method to detect Internet Of Battlefield Things (IoBT) malware via the device's Operational Code (OpCode) sequence. We transmute OpCodes into a vector space and apply a deep Eigenspace learning approach to classify malicious and benign applications. We also demonstrate the robustness of our proposed approach in malware detection and its sustainability against junk code insertion attacks. Lastly, we make available our malware sample on Github, which hopefully will benefit future research efforts (e.g., to facilitate evaluation of future malware detection approaches).

Index Terms—Internet of things malware, internet of battlefield things, malware detection, deep eigenspace learning, deep learning, machine learning

1 Introduction

A typical Internet of Things (IoT) deployment includes a wide pervasive network of (smart) Internet-connected devices, Internet-connected vehicles, embedded systems, sensors, and other devices/systems that autonomously sense, store, transfer and process collected data. In addition to typical civilian settings, IoT has applications in military and adversarial environment. For example in 2017, the U.S. Army Research Laboratory (ARL) "established an Enterprise approach to address the challenges resulting from the Internet of Battlefield Things (IoBT) that couples multi-disciplinary internal research with extramural research and collaborative ventures. ARL intends to establish a new collaborative venture (the IoBT CRA) that seeks to develop the foundations of IoBT in the context of future Army operations".[1]

There are underpinning security and privacy concerns in such an IoT environment. While IoT and IoBT share many of the underpinning cyber security risks (e.g., malware infection), the sensitive nature of IoBT deployment (e.g., military and warfare) makes IoBT architecture and devices more likely to be targeted by cyber criminals. In addition, actors who target IoBT devices and infrastructure are more likely to be state-sponsored, better resourced, and professionally trained.

Intrusion and malware detection and prevention are two active research areas [1], [2], [3], [4], [5]. However, the resource constrained nature of most IoT and IoBT devices and customized operating systems, existing / conventional intrusion and malware detection and prevention solutions are unlikely to be suited for real-world deployment. For example, IoT malware may exploit low-level vulnerabilities present in compromised IoT devices or vulnerabilities specific to certain IoT devices (e.g., Stuxnet, a malware reportedly designed to target nuclear plants, are likely to be 'harmless' to consumer devices such as Android and iOS devices and personal computers). Thus, it is necessary to answer the need for IoT and IoBT specific malware detection.

There has been recent interest in utilizing machine learning and deep learning techniques in malware detection (e.g., distinguishing between malware and benign applications), due to their potential to increase detection accuracy and robustness [1], [6], [7]. Typically, the following criteria are used to evaluate the utility of machine learning and deep learning techniques in malware detection:

- True Positive (TP): indicates that a malware is correctly identified as a malicious application.
- True Negative (TN): indicates that a benign is detected as a non-malicious application correctly.

[1] https://www.arl.army.mil/www/pages/3050/IOBT-Program-Announcement-AmendmentII.pdf, last accessed on February 19th, 2018.

- *A. Azmoodeh is with the Department of Electrical and Computer Engineering, Shiraz University, Shiraz 71345-1978, Iran. E-mail: azmoodeh@cse.shirazu.ac.ir.*
- *A. Dehghantanha is with the Department of Computer Science, University of Sheffield, Sheffield S10 2TN, United Kingdom. E-mail: A.Dehghantanha@sheffield.ac.uk.*
- *K.-K. R. Choo is with the Department of Information Systems and Cyber Security, and Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249. E-mail: raymond.choo@fulbrightmail.org.*

Manuscript received 4 Aug. 2017; revised 25 Dec. 2017; accepted 19 Feb. 2018. Date of publication 27 Feb. 2018; date of current version 6 Mar. 2019.
(Corresponding author: Kim-Kwang Raymond Choo.)
Recommended for acceptance by M. Qiu and S.-Y. Kung.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSUSC.2018.2809665

- False Positive (FP): indicates that a benign is falsely detected as a malicious application.
- False Negative (FN): indicates that a malware is not detected and labeled as a non-malicious application.

Based on the above criteria, the following metrics will then be used to quantify the effectiveness of a given system:

Accuracy is the number of samples that a classifier correctly detects, divided by the number of all malware and goodware (i.e., non-malicious) applications

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{1}$$

Precision is the ratio of predicted malware that are correctly labeled a malware, and is defined as follows:

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

Recall or detection rate is the ratio of malware samples that are correctly predicted, and is defined as follows:

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

F-Measure is the harmonic mean of precision and recall, and is defined as follows:

$$F - Measure = \frac{2 * TP}{2 * TP + FP + FN}. \tag{4}$$

Cross-validation is a fundamental technique in machine learning to assess the extent that the findings of an experiment can be generalized into an independent dataset. While there are many cross validation techniques (e.g., Leave-P-Out, K-fold and Repeated Random Sub-sampling), when the size of a dataset is limited K-fold validation techniques (e.g., 10-fold) are generally used. K-Fold validation techniques are also commonly used to validate the fitness of a model to a hypothetical validation set in the absence of an independent validation set [8], [9].

Due to the fast pace of malware development and the significant increase in the number of malware samples, using deep learning techniques for malware detection is gaining prominence. For example, Yuan et al. [10] used Deep Belief Network on a combination of static and dynamic features of Android APKs and reportedly achieved 96.76 percent of overall accuracy and 95.77 and 97.84 percent of precision and recall respectively in malware detection. Saxe and Berlin [11] presented a detection method using deep feed-forward neural network based on a set of Windows program features (e.g., Byte/Entropy histogram, portable executable (PE) data and metadata and printable character sequence). The authors reportedly achieved detection and false positive rates of 95 and 0.1 percent respectively.

Bilar [12] introduced Operational Codes (OpCodes) as a suitable and reliable feature for malware identification using machine learning techniques. Moskovitch et al. [13] applied different text mining techniques to detect candidate features of Windows malware sample OpCode for classification. The used algorithms such as Neural Network and Decision Tree, and reportedly acheived an accuracy rate of 94.43 percent. Santos et al. [14] used the frequency of a specific OpCode appearance in benign and malicious Windows applications as a feature for Decision Tree, Support Vector Machine, Bayesian Networks and K-Nearest Neighbors algorithms, and reported an accuracy rate of 95.90 percent in malware detection. Hashemi et al. [15] extracted OpCodes of Windows benign and malware executable files, and formed a graph for each sample. Then, they turned the generated graph into a vector using Power Iteration procedure to train classifiers such as Support Vector Machine and Adaboost. They achieved an accuracy rate and a F-measure of 96.09 and 95.98 percent, respectively. Siddiqui et al. [16] utilized occurrence frequency and principle component analysis on a dataset of Windows malware and goodware application OpCodes and reported an accuracy rate of 93.1 percent when they used Random Forest as the classifier.

Thus, we posit that OpCode analysis may provide a basis for a robust and sustainable deep learning based IoT (and IoBT) malware detection system. This is a challenge that has not been addressed in the literature. Specifically, in this paper, we extract OpCode sequence of 1078 benignware and 128 malware (all ARM compatible IoT applications). We utilize Class-Wise Information Gain technique for class aware feature selection [17]. Then, the selected features (OpCodes) of each sample are converted into a graph in which OpCodes are represented by the graph's nodes while the graph edges represent the nodes' affinity in disassembled file of each sample. Finally, generated graphs of malicious and benign samples are used for classification of IoT and IoBT malware and goodware applications using Eigenspace and deep convolutional networks techniques. We achieve 99.68 percent accuracy in detecting malware samples, with precision and recall rates of 98.59 and 98.37 percent respectively.

To the best of our knowledge, this is the first OpCode-based deep learning method for IoT and IoBT malware detection. We then demonstrate the robustness of our proposed approach, against existing OpCode based malware detection systems in [14], [15]. We also demonstrate the effectiveness of our proposed approach against junk-code insertion attacks. Specifically, our proposed approach employs a class-wise feature selection technique to overrule less important OpCodes in order to resist junk-code insertion attacks. Furthermore, we leverage all elements of Eigenspace to increase detection rate and sustainability. Finally, as a secondary contribution, we share a normalized dataset of IoT malware and benign applications,[2] which may be used by fellow researchers to evaluate and benchmark future malware detection approaches. On the other hand, since the proposed method belongs to OpCode based detection category, it could be adaptable for non-IoT platforms.

In the next section, we briefly review related work. Section 3 describes our collection, preprocessing and evaluation methodology. Section 4 presents our proposed approach, followed by its evaluation in Section 5. Section 6 concludes this paper and suggests several future research agenda.

[2] The samples are available on https://github.com/azmoodeh/IoTMalwareDetection, where the benign samples are in binary and the malware samples are in OpCode.

2 Related Literature

Malware detection methods can be static or dynamic [18]. In dynamic malware detection approaches, the program is executed in a controlled environment (e.g., a virtual machine or a sandbox) to collect its behavioral attributes such as required resources, execution path, and requested privilege, in order to classify a program as malware or benign. Static approaches (e.g., signature-based detection, byte-sequence n-gram analysis, opcode sequence identification and control flow graph traversal) statically inspect a program code to detect suspicious applications.

David et al. [19] proposed $Deepsign$ to automatically detect malware using a signature generation method. The latter creates a dataset based on behaviour logs of API calls, registry entries, web searches, port accesses, etc, in a sandbox and then converts logs to a binary vector. They used deep belief network for classification and reportedly achieved 98.6 percent accuracy. In another study, Pascanu et al. [20] proposed a method to model malware execution using natural language modeling. They extracted relevant features using recurrent neural network to predict the next API calls. Then, both logistic regression and multi-layer perceptrons were applied as the classification module on next API call prediction and using history of past events as features. It was reported that 98.3 percent true positive rate and 0.1 percent false positive rate were achieved.

Demme et al. [21] examined the feasibility of building a malware detector in IoT nodes' hardware using performance counters as a learning feature and K-Nearest Neighbor, Decision Tree and Random Forest as classifiers. The reported accuracy rate for different malware family ranges from 25 to 100 percent. Alam et al. [22] applied Random Forest on a dataset of Internet-connected smartphone devices to recognize malicious codes. They executed APKs in an Android emulator and recorded different features such as memory information, permission and network for classification, and evaluated their approach using different tree sizes. Their findings showed that the optimal classifier contains 40 trees, and 0.0171 of mean square root was achieved.

In order to detect crypto-ransomware on Android devices as management nodes of an IoT network, Azmoodeh et al. [23] recorded the power usage of running processes and identified distinguishable local energy consumption patterns for benign applications and ransomware. They broke down the power usage pattern into sub-samples and classified them, as well as aggregating sub-samples' labels to determine the final label. The proposed approach reportedly achieved 92.75 percent accuracy. The need to secure IoT backbone against malware attacks motivated Haddad Pajouh et al. [24] to propose a two-layer dimension reduction and two-tier classification module to detect malicious activities. Specifically, the authors used Principle Component Analysis and Linear Discrimination Analysis to reduce the dataset and then used Naïve Bayes and K-Nearest Neighbor to classify samples. They achieved detection and false alarm rates of 84.86 and 4.86 percent, respectively.

While OpCodes are considered an efficient feature for malware detection, there does not appear to have been any attempt to use OpCodes for IoT and IoBT malware detection. In addition, using deep learning for robust malware detection in IoT networks appears to be another understudied topic. Thus, in this paper, we seek to contribute to this gap by exploring the potential of using OpCodes as features for malware detection with deep Eigenspace learning.

3 Dataset Creation and Feature Selection

We created a dataset of 1078 benign and 128 malware samples for ARM-based IoT applications [25]. All malware samples were collected using VirusTotal[3] Threat Intelligence platform between February 2015 and January 2017. All goodware were collected from a variety of official IoT App stores such as Pi Store.[4]

IoT and IoBT applications are likely to consist of a long sequence of OpCodes, which are instructions to be performed on device processing unit. In order to disassemble samples, we utilized Objdump (GNU binutils version 2.27.90) as a disassembler to extract the OpCodes. Creating n-gram OpCode sequence is a common approach to classify malware based on their disassembled codes [26], [27]. The number of rudimentary features for length N is C^N, where C is the size of instruction set. It is clear that a significant increase in N will result in feature explosion. In addition, decreasing the size of feature increases robustness and effectiveness of detection because ineffective features will affect performance of the machine learning approach.

Therefore, there is a tendency to first apply a feature selection algorithm and find the best features [28] in order to reduce the feature set to avoid feature explosion. Information retrieval techniques are widely used for feature selection [29]. For example, Information Gain (IG) is an information-theoretic approach to select global features by ranking them based on the amount of information content available in a classification problem. IG applies statistical tools to choose global features and does not consider class information. In situations such as imbalanced datasets, global feature selection methods neglect minor class-specified features which may reduce system efficiency.

Class-Wise Information Gain (CIG) [17] is proposed to overcome global feature selection imperfection and aims to recognize more useful features based on available class information. To understand how CIG is calculated for an arbitrary two-class problem, we refer the reader to Equation (5), where $P(v_f=1,C_i)$ denotes the probability of feature f appearing in C_i, and $P(v_f=0,C_j)$ is the probability of feature f being absent from C_j. C_B and C_M denote benign program and malicious applications, respectively.

$$\begin{aligned}CIG(f,C_B)&=P(v_f=1,C_B)*log\frac{P(v_f=1,C_B)}{P(v_f=1)P(C_B)}\\&+P(v_f=0,C_M)*log\frac{P(v_f=0,C_M)}{P(v_f=0)P(C_M)}\\CIG(f,C_M)&=P(v_f=1,C_M)*log\frac{P(v_f=1,C_M)}{P(v_f=1)P(C_M)}\\&+P(v_f=0,C_B)*log\frac{P(v_f=0,C_B)}{P(v_f=0)P(C_B)}.\end{aligned}\tag{5}$$

In this study, 4,543 1-gram and 610,109 2-gram distinct OpCode sequences were extracted, and $CIG(f,C_B)$ and

[3] http://www.virustotal.com

[4] https://thepihut.com/collections/raspberry-pi-store

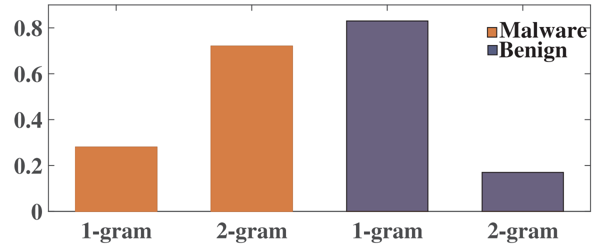

Fig. 1. 1-Gram and 2-gram feature distribution for benign and malware samples.

$CIG(f, C_M)$ were calculated. The top 82 features (approximately 0.01 percent of all features) $\{f_1, \ldots, f_{82}\}$ were selected, where f_i belongs to either the 1-gram or 2-gram category. The size of selected feature set is limited to $j = 82$ because of the significant gap between $f_{j<=82}$'s and $f_{j>=83}$'s CIG values. Due to the high computational resource consumptions, all k-gram($k >= 2$) features were ignored. Features were selected from the 1-gram and 2-gram sequences based on their $CIG(f, C_B)$ and $CIG(f, C_M)$ values. As shown in Fig. 1, majority of the malware features are 2-gram sequences, and 1-gram sequences constitute a large proportion of benign application features. Such knowledge would be crucial in the development of our IoT and IoBT malware detection method, as discussed in the next section.

4 Proposed Approach

Our proposed method is illustrated in Fig. 2, and consists of two phases, namely: OpCode-Sequence Graph Generation phase and Deep Eigensapce Learning phase. Also, feature selection phase is included in Fig. 2.

4.1 Opcode-Sequence Graph Generation

Control Flow Graph (CFG) is a data structure that represents the order of OpCodes in an executable file. A graph, $G = \langle V, E \rangle$, has two sets: V and E. V denotes the graph's vertices and E_{v_i,v_j} shows the relation between V_i and V_j. Previous research has shown the usefulness of this representation in malware detection [14], [15], [30]. $V_i \epsilon \{f_j | j = 1, \ldots, 82\}$ are vertices. and the edges' values represent the relation between vertices (features).

In order to construct the OpCodes' graph, edge values should be computed. The general approach for calculating E_{v_i,v_j} value is to increment E_{v_i,v_j} by 1 when V_i occurs immediately after V_j in the sample's OpCode sequence. Utilizing this procedure would lead to the generation of an adjacency matrix for each sample application within our dataset. Furthermore, normalization of matrix rows would turn E_{v_i,v_j} values into probability of occurrence of V_i. Then, all V_j and E_{v_i,v_j}s values are normalized to a value between 0 and 1. Considering the situations in which V_i and V_j are placed exactly together neglect the longer distance of OpCodes' neighborhood. In other words, merely observing a specific order of OpCodes leads to a crisp representation of OpCode sequence in a graph.

However, the Crisp approach for computing E_{v_i,v_j} has its own drawbacks. Applying feature selection and then incrementing E_{v_i,v_j} by 1 for exact OpCode's occupants results in a sparse adjacency matrix, which may poorly represent a sample file that is not suitable for a classification task. In addition, malware developers may inject useless junk OpCode(s), such as NOP (No Operation) or $(PUSH, POP)$[5] to circumvent/deceive OpCode's neighborhood calculation method.

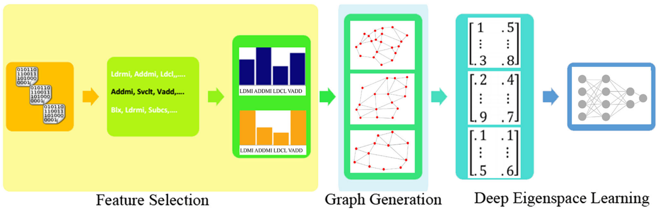
Ldrmi, Addmi, Ldcl,,....
Addmi, Svclt, Vadd,....
Blx, Ldrmi, Subcs,....
LDMI ADDMI LDCL VADD
LDMI ADDMI LDCL VADD
$\begin{bmatrix} 1 & .5 \\ \vdots & \vdots \\ .3 & .8 \end{bmatrix}$
$\begin{bmatrix} .2 & .4 \\ \vdots & \vdots \\ .9 & .7 \end{bmatrix}$
$\begin{bmatrix} .1 & .1 \\ \vdots & \vdots \\ .5 & .6 \end{bmatrix}$
Feature Selection
Graph Generation
Deep Eigenspace Learning

Fig. 2. Proposed approach.

Therefore, we propose a heuristic criteria (see Formulation (6)) to calculate the graph edge values. Fundamental elements of Formulation (6) is the distance between OpCodes. A longer distance increases the divisor exponentially and consequently produces a smaller E_{v_i,v_j}. To improve Formulation (6) by spotting distance mitigates the drawbacks of calculating edges by immediate occurrence and highlights the effect of OpCodes distance. α is a tuning parameter to adjust the impact of OpCode's distance. In this study, we let $\alpha = 1$ has $E_{v_i,v_j} = 1$ for exactly adjacent OpCodes, which is similar to the approach of Hashemi et al. [15]. Also, α can control the effect of OpCodes' distance in detection rate. Formulation (6) would produce a graph of 82 vertices for each given malware and benign sample as the learning material for Deep Eigensapce Learning phase of our method. Algorithm 1 describes graph generation for each sample and Fig. 3 illustrates the output of Opcode-Sequence Graph Generation phase for a sample. For instance, the edge's value between $OpCode_i = call$ and $OpCode_j = sub$ means that $E_{v_i=call,v_j=sub}$ calculated by Formulation (6) is 0.2.

$$E_{v_i,v_j} = \sum_{s \epsilon S} \frac{2}{1 + \alpha * e^{min(|s-t-1|)}}$$

$$\begin{aligned} S = \{&index\ of\ all\ appearance\ of\ OpCode_{V_i} \\ &in\ sample's\ OpCode\ sequence\} \\ t \epsilon \{&index\ of\ all\ appearance\ of\ OpCode_{V_j} \\ &in\ sample's\ OpCode\ sequence\}. \end{aligned} \tag{6}$$

Algorithm 1. Graph Generation Algorithm for Each Sample

```
Input: Sample $P$, Selected Features $F$
Output: Generated Graph for Sample $G$
$k = Number\ Of\ Items\ in\ F$
$G = Zero\ Matrix\ k * k$
for $i = 1$ to $k$ do
    $v_i = F_i$
    for $j = 1$ to $k$ do
        $v_j = F_j$
        $G_{i,j} = E_{v_i,v_j}$ (using Formulation 6)
    end for
end for
$Row\_Normalize\ Matrix\ G$
return $G$
```

[5] After execution of these two OpCodes, the state of program is similar to before executing the PUSH instruction

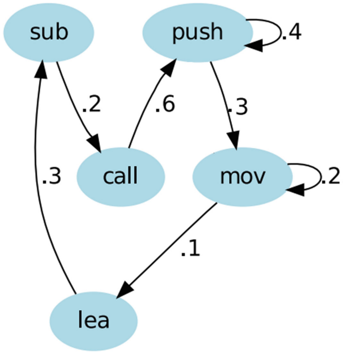

Fig. 3. A schematic view of the sample's generated graph.

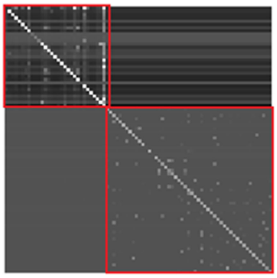

Fig. 4. An Overview of samples' cumulation affinity matrix.

4.2 Deep Eigenspace Learning

4.2.1 Eigenspace

Graphs as a complex data structure for representing relations between vertices are a prevalent data type in machine learning. There are very few data mining and deep learning algorithms [31] that accept a graph as an input [32]. Therefore, a possible alternative is to embed a graph into a vector space [33]. Indeed, graph embedding is a bridge between statistical pattern recognition and graph mining.

Eigenvectors and eigenvalue are two characteristic elements in the graph's spectrum [34], which could linearly transform a graph's adjacency matrix into a vector space (see Equation (7)). v ,λ and A respectively denote eigenvectors, eigenvalues, and a graph's adjacency or an affinity matrix. In this paper, we employ a subset of v and λ for the learning phase.

$$Av = \lambda v. \tag{7}$$

To obtain tangible knowledge of the generated CGFs' structure, a graph that illustrates the cumulative of all samples in our dataset is created (see Fig. 4). Fig. 4 consists of two major diagonal building blocks (marked with red borders), an indication that two main data distributions exist in the given samples. Based on the graph's spectrum theory, in this condition, there should be an explicit eigengap in the matrix's eigenvalues [35], and Fig. 5 depicts the existence of a gap between λ_2 and $\lambda_k (k > 2)$.

Hence, two first eigenvectors of sample's matrix(v_1 and v_2) include much more information about the matrix compared to the remaining eigenvectors, and could represent the whole matrix appropriately.

Moreover, in the learning phase, due to the different data distribution of eigenvalues for malware and benign samples, λ_1 and λ_2 are utilized alongside v_1 and v_2 to detect a sample label and increase our method performance. Figs. 6 and 7 illustrate the difference between malware and benign eigenvalues' ($\lambda_{1,2}$) data distribution, and indicate suitability of employing λ_1 and λ_2 as features for a classification task.

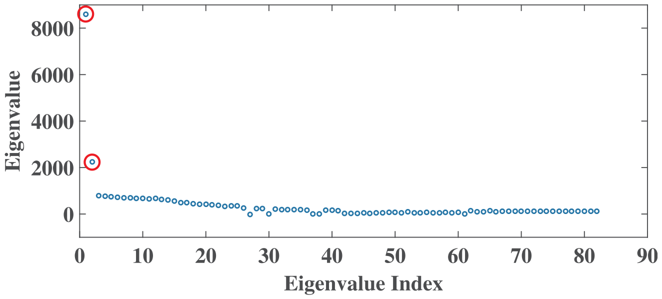

Fig. 5. Sample's cumulation (Fig. 4) eigenvalues.

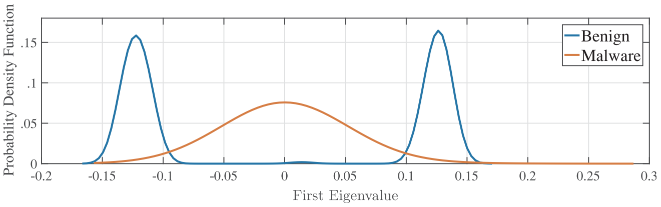

Fig. 6. First eigenvalue (λ_1) distribution.

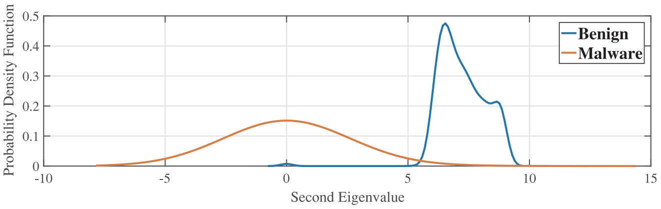

Fig. 7. Second eigenvalue (λ_2) distribution.

4.2.2 Deep Learning

Deep Learning (DL) [36], [37] or deep structured learning is an evolved version of Neural Networks (NN) [38]. A standard NN includes few or many simple, inter-connected nodes called neurons. NN's neurons are organized in a few layers, namely: an input layer, several hidden layers and an output layer. DL focuses on deeper data structure learning by concentrating on the hidden layer's abilities and functionalities. Deep learning has recently been used in a variety of applications, such as speech recognition and machine vision [39], [40]. Variations of DL include Convolutional Networks, Restricted Boltzmann Machines, and Sparse Coding [41].

In this paper, a Convolutional Network is used as the deep learning module of our proposed approach because of its potential for accurate classification in the presence of complex and non-linear data patterns [42]. The first two eigenvectors (v_1 and v_2) and eigenvalues (λ_1 and λ_2) of the samples are used as input values for classification.

5 Findings

In this section, we evaluate the accuracy, precision, recall and F-measure of our proposed approach, in order to demonstrate its robustness in detecting IoT and IoBT malware. Moreover, we demonstrate the sustainability of our proposed approach against junk code insertion attacks.

TABLE 1
Performance Evaluation of Detection Methods on Our Dataset

	Accuracy	Precision	Recall	F-Measure
Proposed Method	99.68%	98.59%	98.37%	98.48%
Hashemi et al. [15]	96.87%	91.09%	81.55%	86.05%
Santos et al. [14]	95.91%	86.25%	77.70%	81.75%

5.1 Robustness

To show the robustness of our proposed approach and benchmark it against existing proposals, two congruent algorithms [14], [15] described in Section 1 are applied on our generated dataset using Adaboost [43] as the classification algorithm. All evaluations were conducted using MATLAB R2015a running on a Microsoft Windows 10 Pro personal computer powered by Intel Core i7 2.67 GHz and 8 GB RAM.

A 10-fold cross validation was used in the validating, and the comparative summary is presented in Table 1. It is clear that our proposed approach outperforms the proposals of Hashemi et al. [15] and Santos et al. [14]. The approach of Santos et al. [14] is a basic and commonly-known OpCode based malware detection algorithm and the approach of Hashemi et al. [15] is the most similar in terms of using eigenspace as the basis.

Accuracy is a general criteria for evaluating performance of an algorithm for both malware and benign class identification. The proposed approach achieves a high accuracy of 99.68 percent, while the approaches of Hashemi et al. [15] and Santos et al. [14] respectively achieve 98.59 and 95.91 percent accuracy. Recall or detection rate is an important criteria and the proposed approach achieves 98.37, in comparison to 81.55 and 77.70 percent for the other two approaches.

Our proposed approach also outperforms the approaches of Hashemi et al. [15] and Santos et al. [14], in terms of precision rate and F-Measure. Utilizing class-wise feature selection appears to result in beneficial features of minor class to be more effective during classification phase. Also, using Formulation (6) to calculate OpCode's distance leads to the ability to represent more OpCode sequence patterns in the sample's graph. It also appears that employing deep neural networks for classification leads to a better classifier.

5.2 Sustainability against Junk Code Insertion Attacks

Junk code injection attack is a malware anti-forensic technique against OpCode inspection. As the name suggests, junk code insertion may include addition of benign OpCode sequences, which do not run in a malware or inclusion of instructions (e.g., NOP) that do not actually make any difference in malware activities. Junk code insertion technique is generally designed to obfuscate malicious OpCode sequences and reduce the 'proportion' of malicious OpCodes in a malware [44].

In our proposed approach, we use an affinity based criteria to mitigate junk OpCode injection anti-forensics technique. Specifically, our feature selection method eliminates less instructive OpCodes to mitigate the effects of injecting junk OpCodes.

To demonstrate the effectiveness of our proposed approach against code insertion attack, in an iterative manner, a specified proportion ({5, 10, 15, 20, 25, 30 percent}) of all elements in each sample's generated graph were selected randomly and their value incremented by one. For example, in the 4th iteration of the evaluations, 20 percent of the indices in each sample's graph were chosen to increment their value by one. In addition, in our evaluations the possibility of a repetitive element selection was included to simulate injecting an OpCode more than once. Incrementing $E_{i,j}$ in the sample's generated graph is equivalent to injecting $OpCode_j$ next to the $OpCode_i$ in a sample's instruction sequence to mislead the detection algorithm. Algorithm 2 describes an iteration of junk code insertion during experiments, and this procedure should repeat for each iteration of k-fold validation.

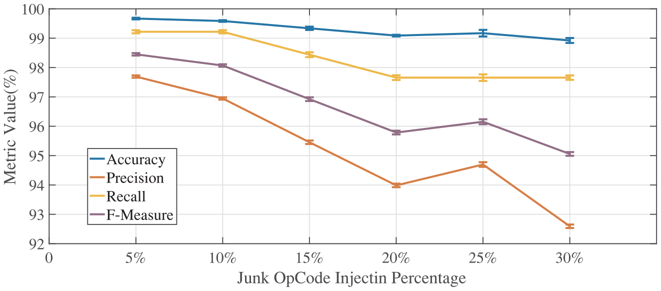

Fig. 8. Evaluation of proposed method robustness over junk code insertion.

Algorithm 2. Junk Code Insertion Procedure

```
Input: Trained Classifier $D$, Test Samples $S$, Junk Code Percentage $k$
Output: Predicted Class for Test Samples $P$
$P = \{\}$
for each $sample$ in $S$ do
    $W$= Compute the CFG of $sample$ based on Section 4.1
    $R$ = {select $k\%$ of $W$'s index randomly(Allow duplicate indices)}
    for each $index$ in $R$ do
        $W_{index} = W_{index} + 1$
    end for
    Normalize $W$
    $e_1, e_2$= 1st and 2nd eigenvectors of $W$
    $l_1, l_2$= 1st and 2nd eigenvalues of $W$
    $P = P \bigcup D(e_1, e_2, l_1, l_2)$
end for
return $P$
```

Fig. 8 reports on the performance of the proposed approach over the stated condition. It is clear that our proposed approach achieves an acceptable performance against junk code insertion attack. A steady trend for Accuracy indicates that truly classified samples significantly outnumber false positive and negative samples—see Equation (1). Furthermore, comparing Precision and Recall results imply that false positive rate or false alarm rate is greater than the false negative rate based on Equations (2) and (3), and taking into account Recall's stable graph, the proposed approach is robust against junk code insertion.

6 Concluding Remarks

IoT, particularly IoBT, will be increasingly important in the foreseeable future. No malware detection solution will be foolproof but we can be certain of the constant race between cyber attackers and cyber defenders. Thus, it is important that we maintain persistent pressure on threat actors.

In this paper, we presented an IoT and IoBT malware detection approach based on class-wise selection of OpCodes sequence as a feature for classification task. A graph of selected features was created for each sample and a deep Eigenspace learning approach was used for malware classification. Our evaluations demonstrated the robustness of our approach in malware detection with an accuracy rate of 98.37 percent and a precision rate of 98.59 percent, as well as the capability to mitigate junk code insertion attacks.

In the future, we plan to evaluate our approach against larger and more diverse datasets, as well as implementing a prototype of the proposed approach in a real-world IoT and IoBT system for evaluation and refinement.

Acknowledgments

The authors would like to thank *virustotal.com* for its sound support for this research. They would also like to thank the editor and anonymous reviewers for their constructive comments. This work was partially supported by the European Council International Incoming Fellowship (FP7-PEOPLE-2013-IIF) grant number 625402.

References

[1] J. Gardiner and S. Nagaraja, "On the security of machine learning in malware c&c detection: A survey," *ACM Comput. Surveys*, vol. 49, no. 3, 2016, Art. no. 59.

[2] J. Peng, K.-K. R. Choo, and H. Ashman, "User profiling in intrusion detection: A review," *J. Netw. Comput. Appl.*, vol. 72, pp. 14–27, 2016.

[3] E. M. Rudd, A. Rozsa, M. Gnther, and T. E. Boult, "A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 2, pp. 1145–1172, Apr.–Jun. 2016.

[4] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, 2017, Art. no. 41.

[5] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh, "Iot security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Serv.-Oriented Comput. Appl.*, Nov. 2014, pp. 230–234.

[6] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 998–1022, Apr.–Jun. 2015.

[7] Z. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2432–2455, Oct.–Dec. 2017.

[8] R. Kohavi, et al., "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 7th Int. Joint Conf. Artif. Intell.*, 1995, pp. 1137–1145.

[9] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," *J. Mach. Learn. Res.*, vol. 5, pp. 1089–1105, Sep. 2004.

[10] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.

[11] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious Unwanted Softw.*, 2015, pp. 11–20.

[12] D. Bilar, "Opcodes as predictor for malware," *Int. J. Electron. Secur. Digit. Forensic*, vol. 1, no. 2, pp. 156–168, Jan. 2007.

[13] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," in *Proc. 1st Eur. Conf. Intell. Security Informat.*, 2008, pp. 204–215.

[14] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inform. Sci.*, vol. 231, pp. 64–82, 2013.

[15] H. Hashemi, A. Azmoodeh, A. Hamzeh, and S. Hashemi, "Graph embedding as a new approach for unknown malware detection," *J. Comput. Virology and Hacking Techn.*, vol. 13, no. 3, pp. 153–166, Aug. 2017.

[16] M. Siddiqui, M. C. Wang, and J. Lee, "Data mining methods for malware detection using instruction sequences," in *Proc. 26th IASTED Int. Conf. Artif. Intell. Appl.*, 2008, pp. 358–363.

[17] Y. Tan, *Class-Wise Information Gain*. Hoboken, NJ, USA: John Wiley & Sons, 2016, pp. 150–172.

[18] K. Shaerpour, A. Dehghantanha, and R. Mahmod, "Trends in android malware detection," *J. Digital Forensics Security Law*, vol. 8, no. 3, 2013, Art. no. 21.

[19] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *Proc. Int. Joint Conf. Neural Netw.*, 2015, pp. 1–8.

[20] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2015, pp. 1916–1920.

[21] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, 2013.

[22] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber Phys. Soc. Comput.*, Aug. 2013, pp. 663–669.

[23] A. Azmoodeh, A. Dehghantanha, M. Conti, and K.-K. R. Choo, "Detecting crypto-ransomware in iot networks based on energy consumption footprint," *J. Ambient Intell. and Humanized Comput.*, pp. 1–12, 2017.

[24] H. H. Pajouh, R. Javidan, R. Khayami, D. Ali, and K. K. R. Choo, "A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks," *IEEE Trans. Emerging Topics Comput.*, to be published, doi: 10.1109/TETC.2016.2633228.

[25] D. Brash, "Recent additions to the armv7-a architecture," in *Proc. IEEE Int. Conf. Comput. Des.*, Oct. 2010, pp. XIX–XIX.

[26] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *J. Comput. Virology*, vol. 2, no. 3, pp. 231–239, 2006.

[27] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informat.*, vol. 1, no. 1, 2012, Art. no. 1.

[28] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digit. Investig.*, vol. 13, no. C, pp. 22–37, Jun. 2015.

[29] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, Mar. 2003.

[30] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *J. Comput. Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[31] D. J. Cook and L. B. Holder, *Mining Graph Data*. Hoboken, NJ, USA: John Wiley & Sons, 2006.

[32] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: John Wiley & Sons, 2012.

[33] K. Riesen and H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding*. Singapore, Singapore: World Scientific, 2010, vol. 77.

[34] F. R. Chung, *Spectral Graph Theory*. Providence, Rhode Island: American Mathematical Society, 1997, no. 92.

[35] M. Newman, "mathematics of networks," in *The New Palgrave Dictionary Econ*. Basingstoke, U.K.: Palgrave Macmillan, 2008.

[36] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[38] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural Networks and Learning Machines*. Pearson Upper Saddle River, NJ, USA: Prentice Hall, 2009, vol. 3.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org

[40] L. Deng, D. Yu, et al., "Deep learning: Methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[41] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognit. Image Anal.*, vol. 26, no. 1, 2016, Art. no. 9.

[42] S. Mallat, "Understanding deep convolutional networks," *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, 2016, Art. no. 20150203.

[43] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Proc. Eur. Conf. Comput. Learn. Theory*, 1995, pp. 23–37.

[44] A. Walenstein and A. Lakhotia, "The software similarity problem in malware analysis," in *Dagstuhl Seminar Proceedings*. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

Amin Azmoodeh received the BS degree in computer engineering and the MSc degree in artificial intelligence from Shiraz University. His main research interests include theory of machine learning and artificial intelligence and its applications especially in cybersecurity and digital forensics. He has several years' experience in analyzing and implementing security mechanism in Enterprise Resource Planning software.

Ali Dehghantanha (SM'17) is a Marie-Curie International Incoming Fellow in Cyber Forensics and a fellow of the United Kingdom Higher Education Academy (HEA). He has served for many years in a variety of research and industrial positions. Other than the PhD degree in cyber security, he holds many professional certificates such as GXPN, GREM, CISM, CISSP, and CCFP. He has served as an expert witness, cyber forensics analysts, and malware researcher with leading players in cyber-security and e-commerce. He is a senior member of the IEEE.

Kim-Kwang Raymond Choo (SM'15) received the PhD degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed professorship with The University of Texas at San Antonio. He serves on the editorial board of *Computers & Electrical Engineering*, *Cluster Computing*, *Digital Investigation*, *IEEE Access*, *IEEE Cloud Computing*, *IEEE Communications Magazine*, *Future Generation Computer Systems*, *Journal of Network and Computer Applications*, *PLoS ONE*, *Soft Computing*, etc. He also served as the special issue guest editor of *ACM Transactions on Embedded Computing Systems* (2017), *ACM Transactions on Internet Technology* (2016), *Digital Investigation* (2016), *Future Generation Computer Systems* (2016 and 2018), *IEEE Cloud Computing* (2015), *IEEE Network* (2016), *IEEE Transactions on Cloud Computing* (2017), *IEEE Transactions on Dependable and Secure Computing* (2017), *Journal of Computer and System Sciences* (2017), *Multimedia Tools and Applications* (2017), *Personal and Ubiquitous Computing* (2017), *Pervasive and Mobile Computing* (2016), *Wireless Personal Communications* (2017), etc. In 2016, he was named the Cybersecurity Educator of the Year APAC (Cybersecurity Excellence Awards are produced in cooperation with the Information Security Community on LinkedIn), and in 2015 he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen-Nuremberg. He is the recipient of the ESORICS 2015 Best Paper Award, 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and the British Computer Society's Wilkes Award in 2008. He is also a fellow of the Australian Computer Society, and a senior member of the IEEE.