Ad Hoc Networks 122 (2021) 102591

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Generative adversarial network to detect unseen Internet of Things malware

Zahra Moti [a], Sattar Hashemi [a], Hadis Karimipour [b], Ali Dehghantanha [c,*], Amir Namavar Jahromi [b], Lida Abdi [a], Fatemeh Alavi [a]

[a] Machine Learning Lab, Department of Computer Science and Engineering and Information Technology, Shiraz University, Shiraz, Iran
[b] Smart Cyber-Physical Systems Lab, School of Engineering, University of Guelph, Guelph, Ontario, Canada
[c] Cyber Science Lab, School of Computer Science, University of Guelph, Guelph, Ontario, Canada

ARTICLE INFO

Keywords:
Malware
Generative Adversarial Network (GAN)
Deeplearning
Convolutional Neural Network (CNN)
Long Short Term Memory (LSTM)
Edge layer
Internet of Things

ABSTRACT

Machine learning is significantly used for malware and adversary detection in the industrial internet of things networks. However, majority of these methods require a significant prior knowledge of malware properties to identify optimal features for malware detection. This is a more significant challenge in IoT environment due to limited availability of malware samples. Some researchers utilized data deformation techniques such as converting malware to images or music to generate features that can be used for malware detection. However, these processes can be time-consuming and require a significant amount of data. This paper proposes MalGan, a framework for detecting and generating new malware samples based on the raw byte code at the edge layer of the Internet of Things (IoT) networks. Convolutional Neural Network (CNN) was utilized to extract high-level features, and boundary-seeking Generative Adversarial Network technique was used to generate new malware samples. Thus, even with a few malware samples, a significant number of previously unseen malware samples are detectable with high accuracy. To capture the short-term and long-term dependency of features, we employed an attention-based model, a combination of CNN and Long Short Term Memory. The attention mechanism improves the model's performance by increasing or decreasing attention to certain parts of the features. The proposed method is examined extensively using standard Windows and IoT malware datasets. The experimental results indicate that our proposed MalGan is the method of choice, as it offers a higher detection rate compared to the previous malware detection algorithms.

1. Introduction

Malicious software, known as malware, is a software that can damage and disrupt a computer systems by encryption, stealing sensitive information, monitoring user activities without their permission, spying, etc. [1]. Malwares are growing in number and complexity. According to McAfee, 100,000 new samples, or about one threat per second, are published daily. Semantic reports that the number of malware in 2018 has increased by about 25 percent [2]. According to the Kaspersky Lab report, 560,025,316 unique URLs were recognized as malicious by Web Anti-Virus components in 2019.

Although there are many techniques developed for malware detection,it is still a popular topic among researchers due to the complexity and increasing numbers of new malwares [3–5]. One of the most traditional methods for detecting malware is signature-based methods. These methods require saving all existing and known malware signatures to detect malware samples [6,7]. The most prominent problem of these methods is that they cannot detect new and unknown malware samples. Also, updating the database with new signatures is time-consuming, and during this time, malware will be able to perform malicious actions [8]. Moreover, storing signatures of all existing malware is very costly and inefficient [9].

heuristic machine learning methods are widely used to overcome these limitations [10–13]. These techniques are first extracting crucial features from malware samples that best describing the content and behavior of the malware [14]. These features are then used to train a model to detect malware samples [15]. However, these methods require identification of important features in advance, which is a costly, time-consuming and sometime impossible due to limited available malware samples which is a significant limit in IoT and IIoT environments [16]. Some researchers tackled this problem by converting the malware to another format, like audio, signal, or image, to find suitable features [14,17–20]. Although some of these approaches have achieved satisfying results [6,20], they still take a significant amount of time to change the data representation and extract the proper features.

* Corresponding author.
E-mail addresses: z.motie@shirazu.ac.ir (Z. Moti), s_hashemi@shirazu.ac.ir (S. Hashemi), hkarimi@uoguelph.ca (H. Karimipour), adehghan@uoguelph.ca (A. Dehghantanha), anamavar@uoguelph.ca (A.N. Jahromi), l-abdi@cse.shirazu.ac.ir (L. Abdi), fatemeh.alavi@shirazu.ac.ir (F. Alavi).

https://doi.org/10.1016/j.adhoc.2021.102591
Received 1 March 2021; Received in revised form 26 April 2021; Accepted 17 June 2021
Available online 15 July 2021

Moreover, depending on the model structure, these approaches may require a significant amount of data for training. As a result, they will require enormous time and powerful hardware such as GPU [21,22].

Currently, Generative model has become one of the most important classes of deep neural networks, which has been considered widely. GAN is a successful generative model introduced in 2014 [23], with two main applications in cybersecurity. First, GAN can improve machine learning models' generalization and performance by generating new samples that resemble the original data. Second, it can be used to generate adversarial examples to attack machine learning models [24]. Li et al. [25] employed GAN to generate adversarial attacks to deceive the Android cloud firewall. They used a modified GAN called bi-objective GAN, which contains two discriminators. The generator's objective is to generate candidate adversarial examples, while the two discriminators' objective is to distinguish malicious from benign and adversarial from normal examples. Thus, the generator will be resistant to firewall and malware detector systems. In another work, Zhang et al. [26] presented a method to detect adversarial examples of cross-site scripting (XSS) attacks. For this purpose, they proposed an improved Monte Carlo Tree Search (MCTS) algorithm whose selection, expansion, and simulation stages are different from normal MCTS. Also, they implemented a GAN-based model to detect adversarial examples. They added adversarial examples to the training set to equip the discriminator for improving detection rate. Li et al. [27] utilized ensemble deep learning to detect and generate adversarial examples and demonstrated its impact on models' performance.

In this paper, we employ deep learning-based algorithms to detect and generate new malware samples. We try to utilize minimal domain knowledge such as file header information. We then use a CNN as a supervised feature learning to learn the conceptual and high-level features from the header's raw bytes and provide more discriminant features. After that, GAN are used to generate the signatures of previously unseen malware samples. Due to the problems and limitations of GAN [23], such as the vanishing gradients, mode collapse, and the instability of the training process, we use boundary-seeking GAN [21]. Since the generated data are based on the distribution of existing malware, they are potentially similar to future malware samples. These synthetic samples are added to the training set. As a result, the model is trained with more diverse samples. Finally, we propose an attention-based deep learning model, a combination of CNN and LSTM. CNN is used to consider short-term dependencies between features; while, LSTM is considered for longer-term dependency. The attention mechanism also improves the model's performance by increasing or decreasing attention to certain parts of the features to improve detection accuracy. The main contributions of this study are as follows:

1. To improve the GAN training stability and enhance the accuracy of classifiers for malware detection, we implement a novel representation learning model based on a CNN to extract a high-level representation of raw data.
2. To defend system security with improving generalization and robustness of models, we oversample the training set by generating the signature of new malware samples using a generative model called Boundary seeking GAN.
3. To accurately detect malware samples, we propose an attention-based deep learning model, a combination of CNN and LSTM. The model considers long-term and short-term dependencies of the input data concurrently and focuses on important features.

We evaluate our models using standard IoT and Windows malware datasets. The remainder of the paper is arranged as follows: Section 2 depicts the related work and, in Section 3, we discuss the fundamental background regarding GAN architecture. In Section 4, the proposed method is introduced. Section 5 reports experimental results, and finally, in Section 6, we summarize and conclude our paper.

2. Literature review

Due to the importance of malware detection, many studies have been conducted so far [13,28–30]. In general, malware analysis techniques are divided into dynamic analysis and static analysis [31].

2.1. Dynamic analysis

In dynamic analysis, files are executed in a virtual or real environment, and decisions are made based on their behavior during execution, such as network traffic, memory usage, system call, and hardware feature. The most notable advantage of this technique is resistance to packing and obfuscation. However, this method is very time-consuming and requires many computing resources; it is not suitable for large datasets. Furthermore, many malware samples recognize the virtual environment and do not show their malicious behavior or show their malicious behavior over a very long time [32].

2.2. Static analysis

Static analysis only collect information about the malware without running it in the virtual environment. Compared to dynamic analysis, it is faster and require less computing resources. In this method, features such as bytecode, opcode and API call system are extracted from the source code to detect malware samples [33]. In this paper, we proposed a malware detection method based on the static analysis; thus, related work in this area are discussed in this section.

Ahmadi et al. [34] proposed a method in which several features have been extracted based on hex view and assembly view, including N-gram, metadata, entropy, string length, symbol, operation code, register, and etc. They presented the classification results based on all the combinations of features, which achieved reasonable results in classifying the malware into its respective class. However, the process of extracting features and training the model is very time-consuming.

Darabian et al. [13] utilized sequential pattern processing techniques to discover the most repeated opcode sequence. Some opcode sequences have appeared frequently in malware samples. Therefore, they calculated the number of the opcode repetitions to encode the opcode sequences accordingly. Finally, using several machine learning algorithms such as Decision Tree, K-Nearest Neighbor (K-NN), RandomForest, multilayer perceptron (MLP), Support Vector Machine (SVM), and Logistic Regression, malicious and non-malicious samples have been classified.

Kumar et al. [35] proposed a method based on the header of files. They considered both some fields of header and some extracted rules based on header property as features. Finally, they used six machine learning algorithms for classification that achieved satisfying accuracy. The feature extraction phase is very time-consuming and inefficient. Some Other presented works tried to solve these problems by changing the representation of files.

Cui et al. [36] proposed a method that first converts malware binary files to gray images. Then, they employed a CNN to classify images to different families. They stated that larger images provides more reliable results. Since the length byte-codes are different, obtaining a fixed size for images is complicated. Besides that, the proposed method comes with a very complicated with large number of inputs, which makes it real implementation impractical.

Gibert et al. [37], also developed a model for malware detection by gray images analysis. They proposed a CNN with three convolution layers and one fully connected layer. After each convolution layer, max-pooling and batch normalization layers were employed which slightly improved the accuracy compared to other similar works in the literature. However, the shortcomings mentioned in the previous work still exist.

Farokhmanesh et al. [19] proposed a malware detection method based on byte codes. In this method, the header of files was extracted

and then converted to MIDI message using a synthesizer. After this conversion, using a music processing method, namely MFCC, proper features were extracted for train machine learning models, including SVM, Random Forest, and KNN. However, determining the optimal value for parameters and converting raw bytes to audio is very time-consuming and inefficient. Besides that, the information in byte code may be more suitable for malware detection.

Raff et al. [38] evaluated the feasibility of minimal domain knowledge for training of neural networks. They extracted 115 features of MS-DOS, File, and Optional header were extracted and then classified them using Random Forest and SVM classifiers. The proposed network includes four fully connected layers following dropout and batch normalization. Their work demonstrates the potential of neural networks to learn from raw byte values, which do not require any particular feature engineering and media conversion as in previous methods.

Most of the previous works for malware detection either convert the original data into media or rely feature engineering. Both of these task are complicated and challenging. Besides that, most of the proposed techniques cannot accurately detect unknown or noisy malware samples. In this work, our goal is to address these problems by elimination the pre-processing steps and generating new malware samples. Consequently, our model can accurately detect majority of the unknown malware accurately.

3. Background

3.1. Generative adversarial learning

The primary goal of the Generative Adversarial Networks is to learn the density function and estimate the distribution of the training set. GAN consists of two neural networks called generator and discriminator, which train with a min–max objective function simultaneously [23]. The generator intends to generate realistic samples to deceive the discriminator while the discriminator tries to correctly classify generated and the real samples. The generator first takes a random vector as input, which contains noise derived from a specific differential function and tries to make it more similar to the real samples in each iteration. The discriminator is an ordinary neural network taking its input samples from two sources: original and generated samples. Then it returns a probability close to one for real data and close to zero for synthetic data. The error rate in the discriminator network is calculated by cross-entropy function and then back-propagated to generator and discriminator networks to update the weights [39]. The objective function of GAN is shown in Eq. (1) where D and G refer to the discriminator and generator, respectively [23]. p_r is the probability distribution of real data, and p_g is the probability distribution of generated data. The goal of the generator is to minimize the objective function, while the goal of the discriminator is to maximize it.

$$\min_G \max_D V(D,G) = E_{x\sim\ p_r}[\log D(x)] + E_{\tilde{x}\sim\ p_g}[\log(1 - D(G(z)))] \tag{1}$$

In the original GAN paper, it is assumed that G is given and the optimal value of the D is equal to [39]:

$$D^*(x) = \frac{p_r}{p_r + p_g} \tag{2}$$

Under an ideal discriminator, despite the traditional maximum likelihood approaches which minimize Kullback–Leibler divergence between p_g and p_r, as indicated in Eq. (3) the generator minimizes the Jensen–Shannon divergence [40] and if p_g is equal to p_r, the global minimum of V(G) is $V^* = -\log 4$ [41].

$$V(G) = 2 * D_{JSD}(P \parallel Q) - \log 4 \tag{3}$$

One of the essential criteria in the convergence of probability distribution is the type of distance used to compare two distributions. This criterion must be continuous and give usable gradients at all points. As mentioned, GAN uses JSD to compare the two distributions, which is one reason for its success [41]. However, some examples show that JSD is not continuous in all points, so it does not offer valuable gradients [42]. There is a fundamental assumption in GAN, which assumes its objective function is a convex–concave.

Nevertheless, it has been proven that even if the generator and the discriminator are both linear, it is concave–concave and not convex–concave, which makes optimization unstable [43]. In theory, it is assumed that the discriminator is totally optimized during the generator update, and then the optimal value for the generator is approximated. In practice, achieving a fully optimized discriminator seems very difficult, and also, the stronger the discriminator, the worse the generator update [42]. GAN suffers from other problems such as mode collapse and vanishing gradient. In mode collapse, the generator learns only small parts of the training data and generates samples without diversity to deceive the discriminator [42]. Many studies have been presented to solve the mentioned problems by changing the generator and discriminator's objective function or structure. One of the proposed models is Boundary-seeking GAN, which was presented by Hjelm et al. [21] in 2017. They generalized the idea of minimizing JSD to another criterion called f-divergence, which Nowozin et al. [44] proposed a gentle formula for it.

Definition 1 (*f-divergence*)**.** Suppose p and q are continuous density functions that provide two distributions of P and Q, respectively, and $f : R_+ \rightarrow R$ is a convex lower semi-continuous function [45].

$$D_f(P \parallel Q) = \int_X q(x) f\left(\frac{p(x)}{q(x)}\right) dx \tag{4}$$

Definition 2 (*Variational Lower-bound of the f-divergence*)**.** Let f^* be a convex conjugate function of f, and $T : X \rightarrow R$ [21].

$$\begin{aligned} D_f(P \parallel Q) &= \int_X q(x) \sup\left\{ t\ \frac{p(x)}{q(x)} - f^*(t) \right\} dx \\ &\geq E_{x\sim p}[T(x)] - E_{x\sim Q}\left[f^*(T(x)\right] \end{aligned} \tag{5}$$

Now, the distribution of P can be estimated using the distribution of Q, with the variational lower-bound of the $f\text{-}divergence$ [44]. According to the Definition 2, we have:

$$D_f\left(P \parallel Q_\theta\right\}) = E_{Q_\theta}\left[f\ \frac{p(x)}{q(x)}\right] = E_{Q_\theta}\left[\sup\left\{ t\ \frac{p(x)}{q(x)} - f^*(t) \right\}\right] \tag{6}$$

When $\frac{\varphi f}{\varphi t}\ (t) = \frac{p(x)}{q(x)}$, Eq. (6) will be maximized. Reword t as $T(x)$ as a discriminator, which $T^*(x)$ is its optimal value in Eq. (6). So the target distribution is equal to:

$$p(x) = q_\theta \frac{\partial f^*}{\partial T}\left(T^*(x)\right) \tag{7}$$

Therefore, the target distribution is obtained based on a coefficient of the generated distribution function, which is considered as an optimal weight for relevant samples. The optimal weight in the original paper of GAN is considered as follows [21]:

$$w^*(x) = \frac{\partial f^*}{\partial T}\left(T^*(x)\right) = \frac{D^*(x)}{1 - D^*(x)} \tag{8}$$

However, achieving the optimal discriminator is challenging, so the target density function is estimated using a sub-optimal discriminator, which determines a lower-bound of $f\text{-}divergence$. As a result, the generator function has been converted to a concave function, and the training process will be more stable. Target function of the generator is defined as follows [21]:

$$\min_g E_q\left[\frac{1}{2}(\log T(x) - \log(1 - T(x)))^2\right] \tag{9}$$

4. Proposed method

4.1. Overview of the proposed method

Generally, instead of creating a program from scratch, malware developers produce new samples by making minor changes to existing

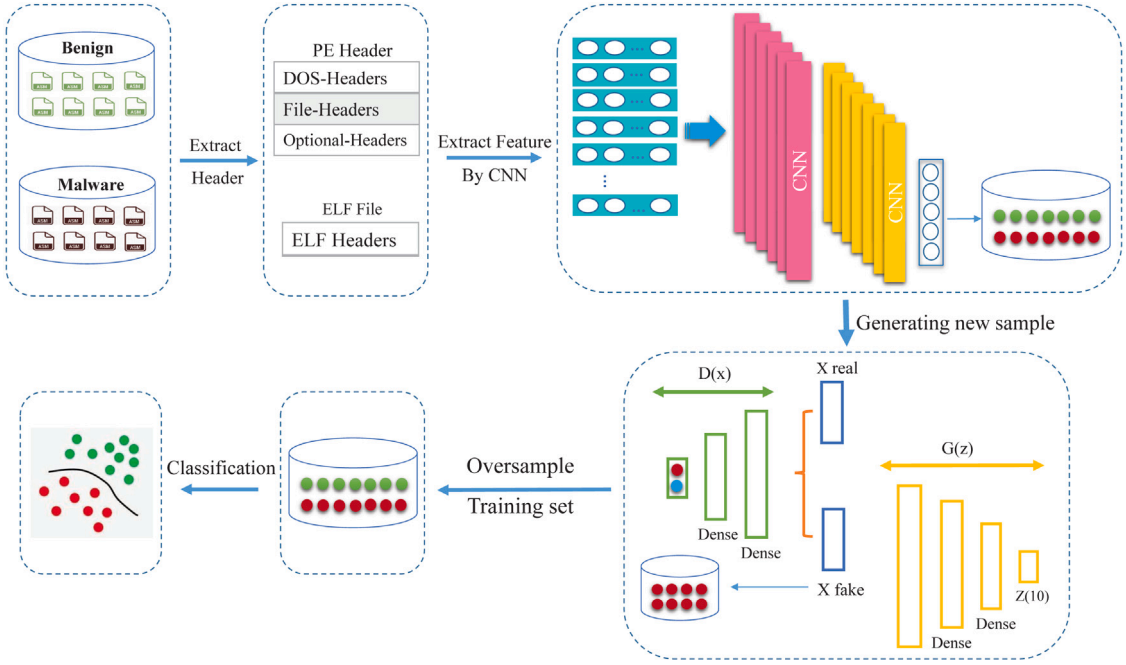

Fig. 1. Framework of the proposed method.

ones for make the detection harder. In this paper, we generated new malware signatures based on a limited number of existing ones using GAN. Since the ultimate objective of GAN is to estimate the distribution of data, the generated data are based on the distribution of existing malware; they are potentially similar to malware that may be released in the future. After generating the new samples, we add this data to the training set, which causes the model to be trained with more diverse samples.

Fig. 1 shows the structure of our proposed framework, MalGan. First, headers of the executable files are extracted, then the high-level and abstract features are derived using the CNN. After that, new signature of malware samples are generated by boundary-seeking GAN and added to the original dataset to train our proposed deep network.

4.2. Header extraction

We utilize raw byte sequences of header as a feature to detect malware samples. The header is a small part of the file containing beneficial information, which can be extracted under any circumstances because it is not encrypted and compressed [19]. Besides, we tried to use the minimum domain of knowledge, we utilized only three parts of the PE header, including MS-DOS, FILE, and OPTIONAL header. Based on several performed experiments on raw bytes and analyzed other studies [38,46,47], we concluded these three header fields have the most significant impact on malware detection in PE files. These three sections are located in most PE files and can be easily extracted with minimal overhead, and the length of the header and the location of the offset is just needed for extraction. The time required to extract these headers is 0.02 s, which is very short compared to extracting features such as opcode. According to PE specifications, the MS-DOS header is the first 64 bytes of a PE file. Information about FILE and OPTIONAL is listed in the next 248 to 264 bytes, depending on whether the program is for 32-bit or 64-bit devices [38]. Also, in the Executable and Linking Format (ELF) files, we extracted the ELF header, which depends on the 64-bit or 32-bit address, containing the initial 64 or 52 bytes of the file [48]. Each extracted byte is converted to a decimal number between 0 and 255 and used as input of neural network in the next step. An ELF header is a roadmap to illustrate the file structure, which includes beneficial information such as object file type, required architecture, version, and etc. We try to use only a small parts of an executable binary file in this work, and there is no need for analysis or extraction. The required time to extract ELF header is about 0.0001 s, which is inconsiderable.

4.3. Extracting high-level features

One application of deep learning is the ability to learn conceptual features automatically. They take a low-level representation of data and extract a high-level representation which is more separable for classification purposes. In other words, conceptual space is learned from the original space. Another advantage is dimensionality reduction. The larger the dimensions of the feature space, the more memory is required to perform the calculations. In this paper, we intended to implement a deep representation learning model. Thus, we tested two of the most popular models, including Autoencoder and CNN, widely utilized for feature extraction. According to our experiments, which are presented in the following section, CNN was superior to autoencoder for feature extraction. In our proposed deep network, first the Embedding layer is used to convert the discrete vectors of features to continuous vectors [49]. There are other ways to do this, including one-hot encoding. In one-hot encoding, creating and updating a vector is fast and straightforward. However, in some cases, its dimensions may increase dramatically, and it cannot learn the relationship between features vector. Nevertheless, the embedding method can learn logical connections between feature vectors. Then, two 1-dimensional convolutional(Conv-1D) layers are employed, which can extract dependencies between the features. Due to its structure, Conv-1D has a much lower computational complexity than Conv-2D and can be trained at high speeds without any powerful hardware. As a result, it can be used in low-cost and real-time applications. Another advantage of Conv-1D is that even with a shallow architecture can learn complex structures of sequential data [50]. After each convolutional layer, a max-pooling layer is added, which is a standard method for down-sampling. Pooling is responsible for collecting beneficial and abstract features to prevent overfitting and reduce computational complexity [49]. Finally, a dense layer with 32 neurons is added, which is the vector of the newly extracted features.

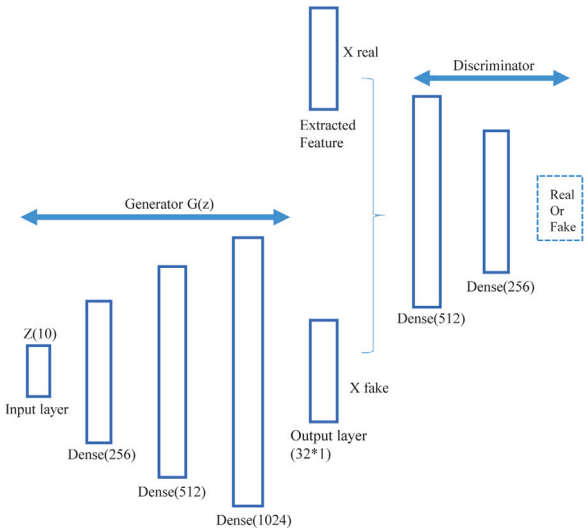

Fig. 2. The general structure of the generator and discriminator.

4.4. Data generation and oversampling the training set

We utilize Boundary Seeking GAN to generate signatures of the new malware samples. The generator generates samples based on a random distribution such as Gaussian. It tries to transfer this random distribution to the distribution of original malware samples. Then, generated malware samples and real ones are given to the discriminator to be classified into the relevant classes. After training the networks, generator learns the distribution of the original malware and will able to generate a new signature of malware samples. Finally, the generated data are added to the original dataset. Our model is more robust to noise and some other modifications and tampering in data due to this oversampling. The structure of the generator and discriminator networks is shown in Fig. 2. Since we utilize high-level features, which CNN extracts in the previous step as training samples, the training process is much faster and more stable than when the raw features are used. Also, we apply a one-side label smoothing technique and change the positive class label by 0.9. According to Goodfellow [39], this technique improves the performance of the discriminator against generator tricks.

4.5. Malware detection

Finally, we proposed a deep network to detect malware, which its overall structure is depicted in Fig. 3. First, there are two one-dimensional convolutional layers that extract several local features from the long input sequences. Subsequent feature maps are calculated based on the following formula, in which the input is indicated by f with length of n and kernel by h with length of m.

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2) \tag{10}$$

After each convolutional layer, there are pooling and dropout layers to prevent overfitting [49]. Dropout is a technique that accidentally removed some neurons of each layer at a constant rate during the network training, so the model does not depend on a specific set of neurons and weights. Then, an LSTM layer is added, which reduces the effect of vanishing gradient and paves the way for working with long sequences [51]. LSTM contains inner mechanisms named gates that can adjust the flow of information.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right) \tag{11}$$

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right) \tag{12}$$

$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right) \tag{13}$$

$$C_t = f_t * C_{t-1} + i_t * \tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_C\right) \tag{14}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{15}$$

In the above equations, f_t denotes the forget gate, which is responsible for removing information no longer needed from the state cell. i_t represents the input gate that decides what new information can be added to the cell, and i_t indicates the output gate that specifies the output of the current cell. C_t represents the cell state, and h_t denotes the hidden state, which is the output of LSTM layer.

To reduce the execution time, the convolutional and max-pooling layers have been used to extract important features from long sequences and pass them to LSTM layer. The combination of CNN and LSTM captures short-term, and long-term dependencies concurrently [52]. In other words, CNN extracts local features based on the filter that convolves on input and LSTM with multiple gates, and memory considers correlations between the features. Then, attention mechanism [53] is used to determine the relevance between different positions of a sequence. The motivation is to enhance accuracy by increasing or decreasing the concentration and contribution of certain parts of the input sequences. The attention mechanism imitates the mechanism of human vision. When the human sight mechanism identifies an item, it will not scan the entire scene, and it will concentrate on a specific part according to the target. Attention learns the correlation between features and their impact on target identification by assigning scores to features. The steps are explained in the following equations. h_i indicates the latent states of the LSTM layer, w_0 and w_1 also refer to the matrix of weights. At first, a neural network is trained to assign scores to connected and interconnected inputs. When scores are calculated, attention weights are generated by applying the softmax function. The summation of the weights should be one. This vector is then multiplied by the input vector to apply the weights to the output. Here, the input is the hidden state of the LSTM layer, so the weight is calculated for each state, and it indicates its importance for a classification task and makes the model more robust against noise and outliers. The output of this layer (out) is given to a fully connected layer with the softmax activation function for classification.

$$\text{out} = \sum_{i=1}^{T} \alpha_i h_i \tag{16}$$

$$\bar{h} = \frac{1}{T} \sum_{i=1}^{T} \alpha_i h_i \tag{17}$$

$$\alpha_i = \tan\left(w_0^T h_i + w_1^T \bar{h}\right) \tag{18}$$

5. Experimental setup

A system with Intel $Core^{TM}$ i7−6700K CPU and 16.0 GB of RAM with Ubuntu 18.04 is used for our experiments. All of the neural networks in the proposed model are built using the Python 3.6 and Keras library. Adam optimizer is utilized, and the batch size is set to 80. Loss function is also set to binary cross-entropy. CNN has been trained for 15 epochs as a feature extraction module, and BGAN is trained for 1200 epochs to generate a new signature of malware samples. More detail about discriminator and generator networks are shown in Tables 1 and 2. Also, our deep network for malware detection is trained for 20 epochs.

5.1. Dataset

We use two Windows malware datasets to evaluate our proposed method. The first dataset contains 5,000 executable files, including 2,500 malware samples and 2,500 non-malicious files. Malware samples are collected from a collection of virusShare online malware repository. Benign samples are gathered from system files of an original

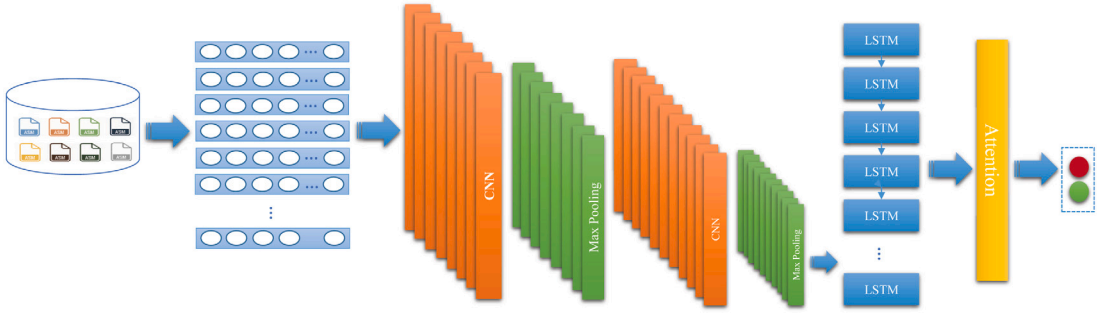

Fig. 3. Structure of designed network to detect malware.

Table 1
Generator structure.

Layer(type)	# Neuron	Activation
Dense	256	Leaky Relu
Batch normalization	–	–
Drop out	–	–
Dense	512	Leaky Relu
Batch normalization	–	–
Drop out	–	–
Dense	1024	Leaky Relu
Batch normalization	–	–
Drop out	–	–
Dense	32	Tanh

Table 2
Discriminator structure.

Layer(type)	# Neuron	Activation
Dense	512	Leaky Relu
Drop out	–	–
Dense	256	Leaky Relu
Drop out	–	–
Dense	128	Leaky Relu
Drop out	–	–
Dense	64	Leaky Relu
Dense	1	Sigmoid

Windows 10 and other known programs, checked with a commercial anti-virus to assure that they are not malicious. The second dataset includes 15,000 EXE and DLL files, of which 7,500 samples as malware, randomly selected from the VX-Heaven dataset, including various malware types, including trojan, worm, backdoor, virus, and hack-tool, and 7,500 samples as non-malicious software. Also, we employ an IoT dataset, which includes 271 malware samples and 281 malicious samples. The last dataset contains 4,200 executable files, which malware samples are randomly selected from the VX-Heaven dataset and packed using a popular tool called UPX. The purpose of selecting this dataset is to examine the selected feature's ability to detect malware in packed mode. We summarize the information of all datasets in Table 3.

5.2. Evaluation metrics

The main criteria for evaluating the performance of the algorithms are as follows:

- True Positive (TP) donates the positive class samples that are correctly classified.
- True Negative (TN) donates the negative class samples that are correctly classified.
- False Positive (FP) donates the negative class samples that are mistakenly classified as positive class samples.
- False Negative (FN) denotes the positive class samples that are mistakenly classified as negative class samples.

Using these measures, AUC, accuracy, precision, recall, and F1. These evaluation metrics are presented in Eqs. (19) through (22), respectively.

- AUC is the probability that the classifier rank a randomly taken positive sample higher than a randomly chosen negative instance. Its numerical value is between zero and one, and the closer it is to one, the better the model performed.
- Accuracy: indicates the ratio of correctly predicted samples to the total samples.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

- Precision: indicates the ratio of correctly predicted malware to total predicted malware.

$$\text{Precision} = \frac{TP}{TP + P} \tag{20}$$

- Sensitivity or Recall: indicates the ratio of predicted malware to the total malware sample in the dataset.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{21}$$

- F1 is the weighted average of precision and recall.

$$\text{F1 score} = \frac{2 * (\text{Recall} * \text{Precision})}{\text{Recall} + \text{precision}} \tag{22}$$

6. Results and discussions

6.1. Validation of generated samples by BGAN

1. Loss function evaluation
   We first plotted the loss values of BGAN to evaluate the effect of our extracted features by CNN training and to examine how the generator and discriminator converged. As shown in Fig. 4, when we used raw byte codes to train BGAN, the error between generator and discriminator is not balanced and tends to vary rapidly. However, as illustrated in Fig. 5, the error rate is balanced using the extracted features, and the error values do not change quickly.
2. Display the distribution of data in two dimensions
   To provide an intuitive understanding of the distribution of real, generated, and non-malicious data distribution, we have displayed these samples in two dimensions using supervised principal component analysis [54]. The blue dots indicate generated data, and the red and green dots represent real malware and benign sample, respectively. As it can be seen from Fig. 6, the boundary-seeking GAN has learned distribution of the real data.
3. Applying statistical measure to compare the similarity of two distributions
   Applying statistical measure to compare the similarity of two distributions To show BGAN has learned the distribution of

Table 3
Dataset description.

Dataset	# of classes	# of malicious files	# of non-malicious files
Collected dataset	2	2500	2500
VX-Heaven dataset	2	7500	7500
IoT dataset	2	271	281
Packed dataset	2	2100	2100

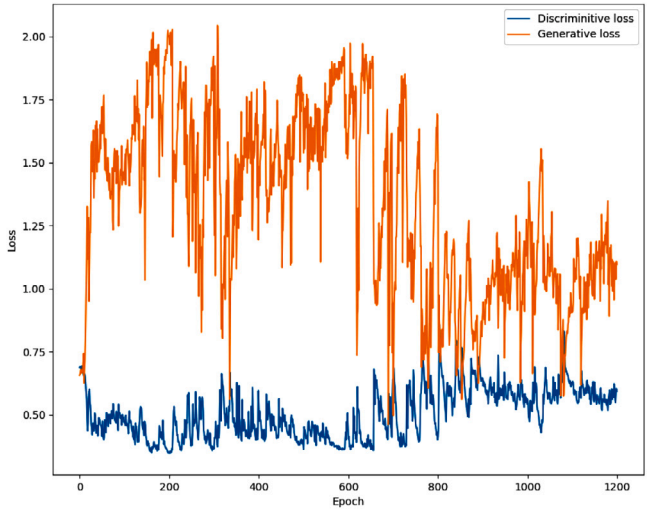

Fig. 4. Generator and discriminator loss when raw header is used to train the BGAN.

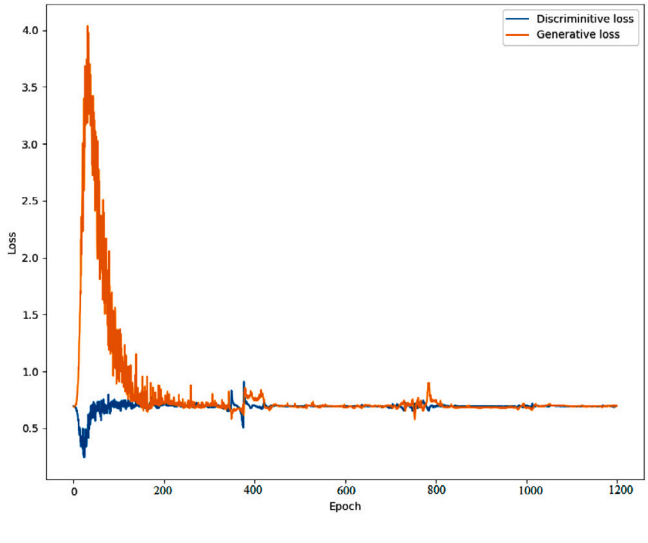

Fig. 5. Generator and discriminator loss when CNN features are used to train BGAN.

Table 4
Comparison between the distribution of original VX-Heaven and synthetic sample generated by our proposed framework.

MMDl	JSD	KLD (fake and real)	KLD (real and fake)
0.054	0.007	0.017	0.016

malware samples well, we have used some popular statistical measures, which calculate the similarity between two distributions, including Kullback–Leibler divergence (KLD) [44], Jensen–Shannon divergence (JSD) [40], and Maximum Mean discrepancy (MMD) [55]. As shown in Table 4, these criteria are close to zero, which means the generated data distribution is very close to the real one.

4. Comparison with SMOTE method

To further evaluate the proposed method's performance, we compare the generated data by BGAN with one of the most popular oversampling methods, the Synthetic Minority Over-Sampling Technique (SMOTE) [56]. We compared the effect of generated data by both methods on several machine learning algorithms. The results are shown in Table 5, which indicate oversampling by BGAN has a significant impact on the improvement of classifiers.

5. Use an external discriminator
To evaluate the generator's performance and the internal discriminator, we used another classifier to distinguish between generated and real samples. Inspired by the proposed method in [53], we used the 1-Nearest Neighbors classifier for this purpose. Ideally, The accuracy of this classifier for distinguishing real and artificial samples should be about 50%, and any value less than this is an indication of overfitting. Our experiment resulted in an accuracy of 57%, which verifies the proper performance of BGAN.

6.2. Computational complexity

Our proposed framework contains pre-processing and malware detection phases. The pre-processing phase includes the header extraction, the high-level feature extraction by CNN, and data generation by boundary seeking GAN processes. Since data generation is performed offline, it is just considered in computational complexities of training. Computational complexity is calculated for the worst-case scenario, in which the number of layers, number of neurons, hidden units, and filters in each layer is supposed to be n, which is the number of training samples. The header extraction step consists of an iterative loop, which in the worst case is of order $O(n)$. According to the algorithm of GAN [23], considering the worst case for the number of training iterations as well as the number of steps to apply to the generator and discriminator, which are both composed of dense layers with the complexity of $O(n^2)$ [57], the computational complexity of GAN is $O(n^2)$. A CNN model was used to extract high-level features. As mentioned in [58], the computational cost of the entire Convolutional layers per training epoch is $O((\sum_{l=1}^{n_{cnn}} n_{l-1} s_l^2 n_l m_l^2) len_i)$, where n_{cnn} is the number of convolutional layers, n_l is the number of filters in the lth layer, n_{l-1} is the number of $l-th$ layer's input channels, s_l is the filter's spatial size, m_l is the spatial size of the output feature map, and len_i is the length of the input data. We generalize this phase to the worst-case; therefore, the order of this algorithm will be $O(n^3)$ for training and $O(n^2)$ for testing computational complexity. Thus, the overall computational complexity of the pre-processing phase in the worst scenario is $O(n^3)$ and $O(n^2)$ for training and testing . For malware detection, we presented a novel technique of an attention-based deep learning model integrating CNN and LSTM for Malware detection. As discussed in [59], the computational complexity of LSTM per epoch is $O(wlen_i)$, where w is the number of weights, and $len_{i'}$ is the input data length, which is generalized to $O(n^4)$ considering the worst-case scenario for training and generalized to $O(n^2)$ for testing computational complexity. The computational cost of the attention layer is $O(n)$ [53]. In total, the computational complexity of Malware detection is the order of $O(n^3) + O(n^4) + O(n) = O(n^4)$ and $O(n^2) + O(n^2) + O(1) = O(n^2)$ for training and testing respectively. Hence, the overall computational complexity of training and testing, which contains pre-processing and malware detection phases, is $O(n^3) + O(n^4) = O(n^4)$ and $O(n^2) + O(n^2) = O(n^2)$.

Table 5
Performance comparison between the proposed method and the SMOTE algorithm.

Methods	SMOTE		MalGan	
	Accuracy	F1	Accuracy	F1
Decision tree	89.65	89.88	93.12	93.21
SVM	89.25	89.07	93.25	93.25
Logistic regression	87.67	87.32	93.50	98.95
3-nn	88.12	88.63	92.56	92.60

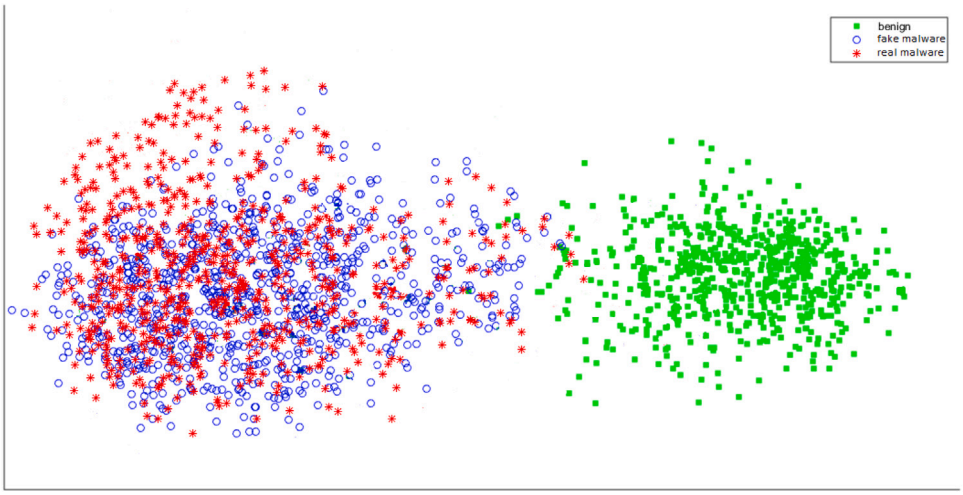

Fig. 6. Illustration of the distribution of the real and generated malware by BGAN and non-malicious files.

6.3. Malware detection performance

6.3.1. Evaluation of the proposed deep network

We compared the performance of the proposed deep network with several machine learning algorithms, including KNN, Random forest, Decision trees, Support Vector Machine (SVM) with linear kernel, and Logistic Regression, which were set to the default parameter in the Scikit-learn library. Also, we compared our model to the standard MLP containing three-hidden layers, LSTM comprising two-hidden layers, and CNN with the same structure as the proposed CNN used as a feature extraction module. CNN and LSTM are among the most popular deep learning models that have gratifying performance in various fields, so it is necessary to compare the proposed deep network with them. For this purpose, we randomly selected malware samples from VX-Heaven dataset. We then compared the proposed deep network with some classical machine learning models and some neural networks. As shown in Fig. 7, the accuracy of the CNN-LSTM model is higher than that of CNN and LSTM, so it can be concluded that short-term and long-term dependencies are explored simultaneously by combining CNN and LSTM models, and the accuracy of the model is improved. Also, our proposed deep network has a better performance than the CNN-LSTM model. It is because of the effect of the attention mechanism on the model. As we said, using the attention mechanism, the essential features for malware detection gain more significant weight, which increase the accuracy of the proposed model. Also, based on [53], in terms of computational complexity, attention layers are faster than convolutional and recurrent layers.

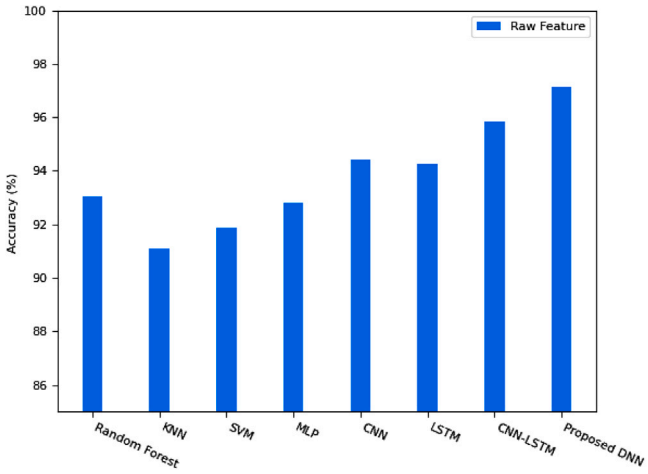

Fig. 7. Compare the accuracy of our proposed deep network and other models.

6.3.2. Detecting packed malware

We performed a test on the forth (packed) dataset to verify the ability of the file header as a static feature to detect malware in packaged mode. All files are packaged using the well-known tool called UPX. As illustrated in Fig. 8, we compared packed and non-packed mode; in the packaged mode, we have achieved acceptable results in detecting malware, which indicates the ability of the header to detect them, even if the files are packed.

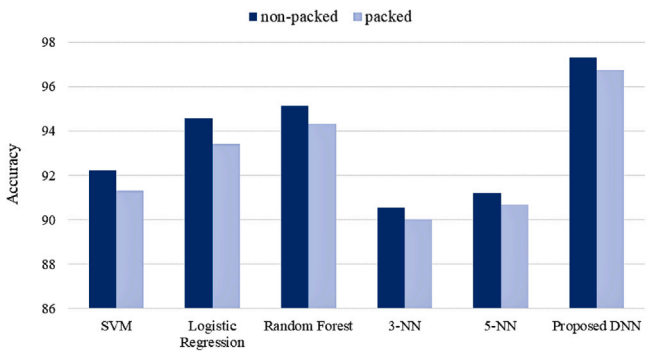

Fig. 8. Comparison between packed and non-packed modes, considering the raw header as a feature.

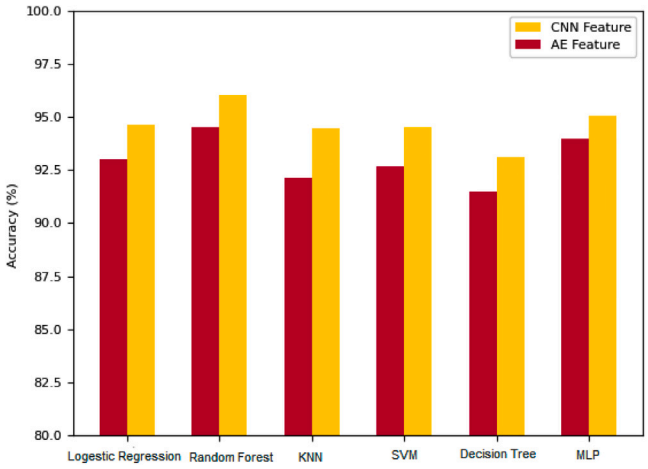

Fig. 9. Comparison between CNN and Autoencoder as a feature extraction.

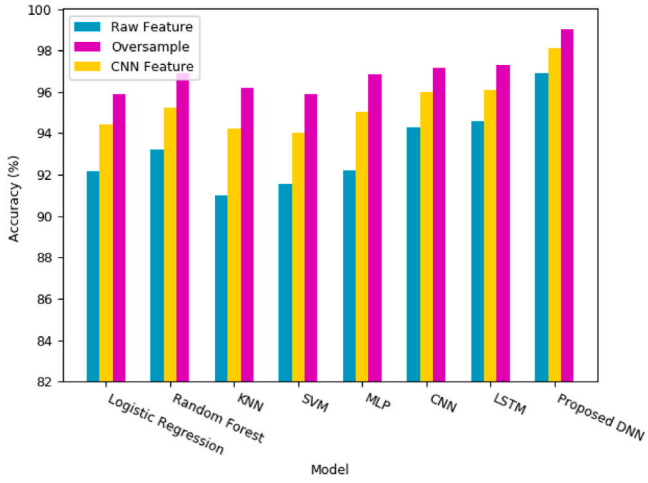

Fig. 10. Accuracy of models on VX-Heaven dataset.

6.3.3. Experiment results on high-level feature

We tested two of the most popular models, including Autoencoder and CNN, widely utilized for feature extraction. We randomly picked malware samples from VX-Heaven dataset. We compare our proposed feature extraction model with a deep Autoencoder containing convolution and fully connected layers. Autoencoder is composed of two main parts: an encoder that maps the input into codes or feature maps and a decoder that maps the codes to a reconstruction of the input. After the models were trained, we used the extracted features to detect malware using several machine learning models. As shown in Fig. 9, the accuracy of all classifiers were trained using CNN-extracted features is better than that of the autoencoder. According to our experiments, CNN was superior to autoencoder for feature extraction.

6.3.4. Evaluation of MalGan

Finally, we examined the effect of over-sampling using BGAN to improve the models' performance. To perform the experiments, we employ a limited number of samples to train the models and finally detect a vast number of new malware samples. We randomly pick 40% of all data for training and the other 60% for testing in VX-Heaven and our collected dataset. We also use 50% of the IoT dataset samples to train the models and the remainder for evaluation. Figs. 10 through 21 demonstrate the result of our experiments. Based on the results, it is clear that the performance of the models has been significantly improved after applying oversampling by BGAN. We oversample the

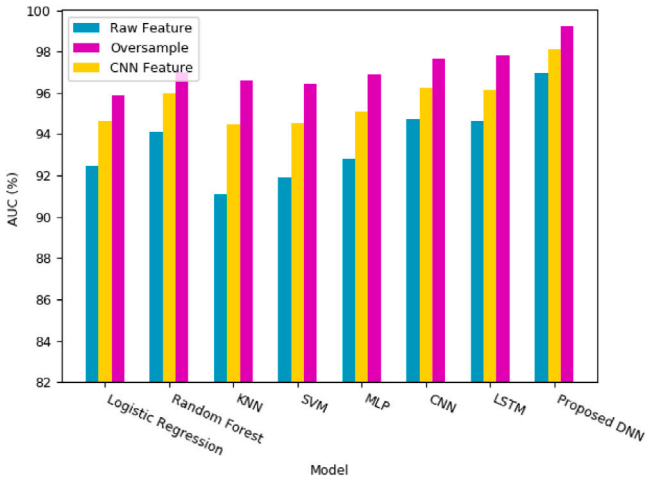

Fig. 11. AUC of models on VX-Heaven dataset.

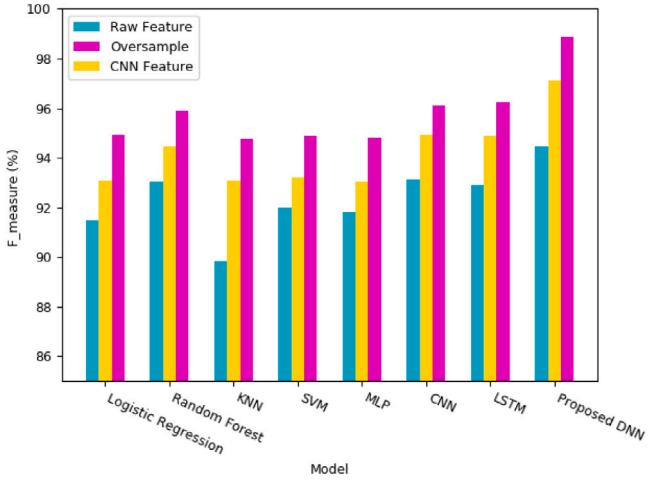

Fig. 12. F1 score of models on VX-Heaven dataset.

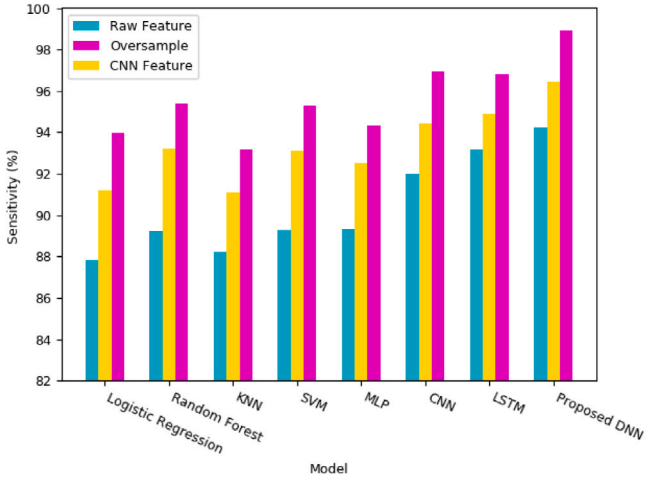

Fig. 13. Sensitivity of models on VX-Heaven dataset.

positive class (malware) using the generated samples by BGAN, and the negative class (benign) using the original software of windows. First, we increased the training data by 50% of the training samples, then by 70% and finally by 100%. It is obvious that the performance

Table 6
Performance comparison of different techniques on VX-Heaven data set.

Model	Accuracy	F1	Train time (m)	Test time (s)
Gibert et al. [37]	90.34±0.60	90.21±0.65	1800	60
Kumar et al. [35]	92.21±0.48	92.53±0.45	54	1.9
Farrokhmanesh et al. [19]	86.45±0.78	85.24±0.75	150	5
Le et al. [60]	92.33±0.45	92.56±0.40	20	0.16
MalGan	**99.03±0.35**	**98.85±0.38**	**5**	**0.004**

Table 7
Performance comparison of different techniques on collected data set.

Model	Accuracy	F1	Train time (m)	Test time (s)
Gibert et al. [37]	84.25±0.45	83.55±0.45	450	60
Kumar et al. [35]	89.55±0.30	89.84±0.32	12.5	1.9
Farrokhmanesh et al. [19]	79.20±0.50	78.57±0.55	37.5	5
Le [60]	89.86±0.21	90.10±0.42	10	0.16
MalGan	**97.56±0.35**	**97.61±0.35**	**2.5**	**0.004**

Table 8
Performance comparison of different techniques on IoT data set.

Model	Accuracy	F1	Train time (m)	Test time (s)
Gibert et al. [37]	96.65±0.50	96.50±0.50	300	60
Namavar et al. [30]	99.56±0.20	–	**0.018**	**0.0009**
Le et al. [60]	98.54±0.10	98.65±0.25	5	0.16
Haddadpajouh et al. [11]	98.18	–	–	–
MalGan	**99.96±0.12**	**99.97±0.03**	1.5	0.003

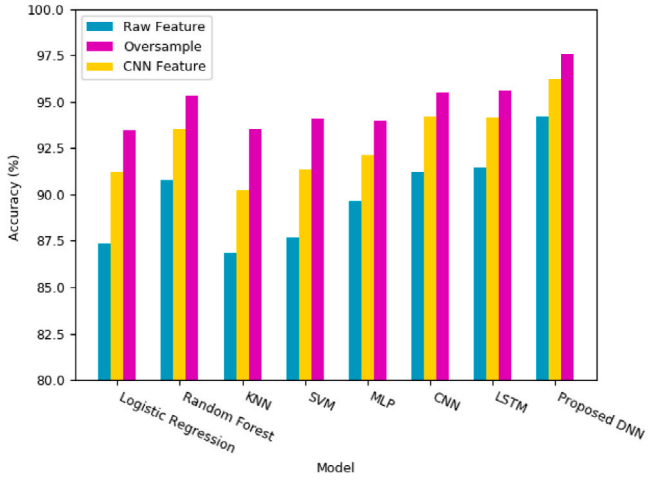

Fig. 14. Accuracy of models on Collected dataset.

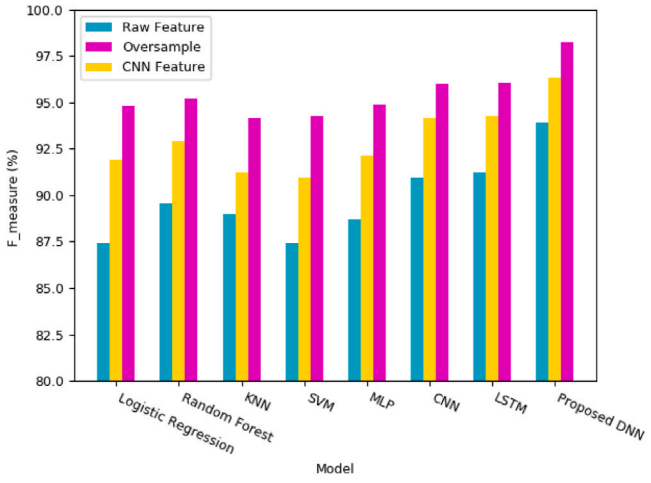

Fig. 16. F1 score of models on Collected dataset.

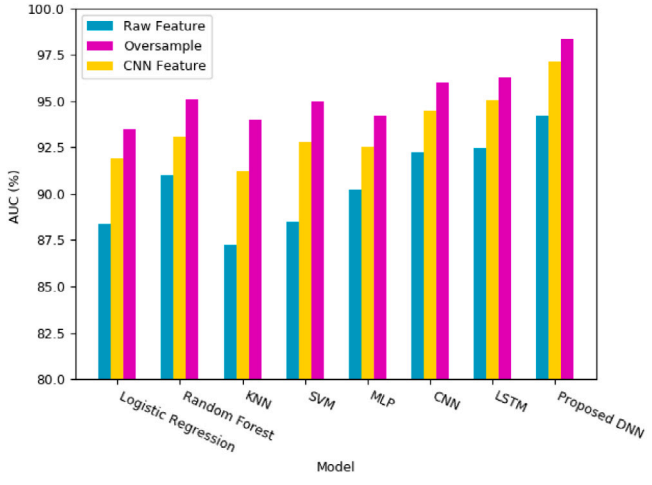

Fig. 15. AUC of models on Collected dataset.

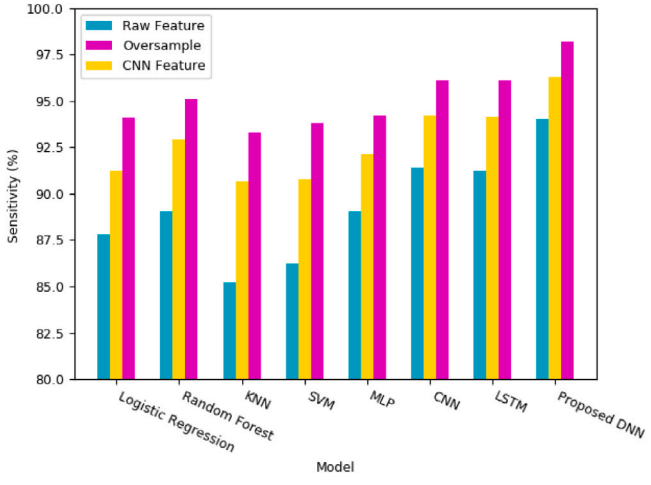

Fig. 17. Sensitivity of models on Collected dataset.

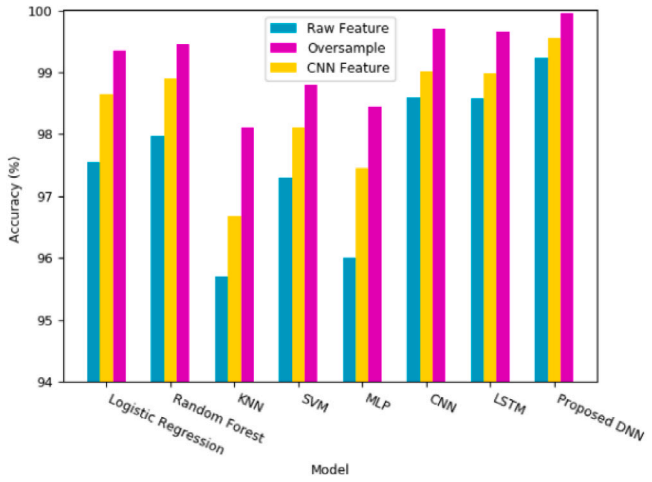

Fig. 18. Accuracy of models on IoT dataset.

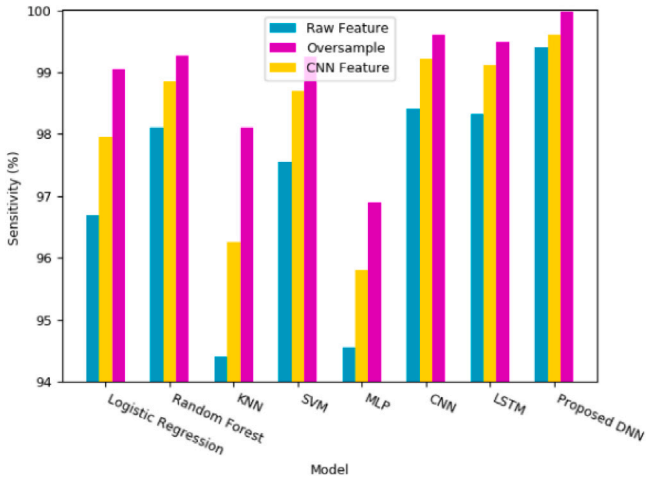

Fig. 21. Sensitivity of models on IoT dataset.

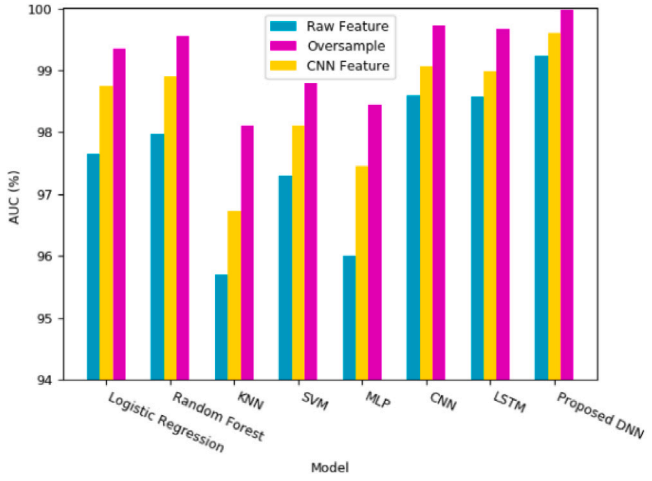

Fig. 19. AUC of models on IoT dataset.

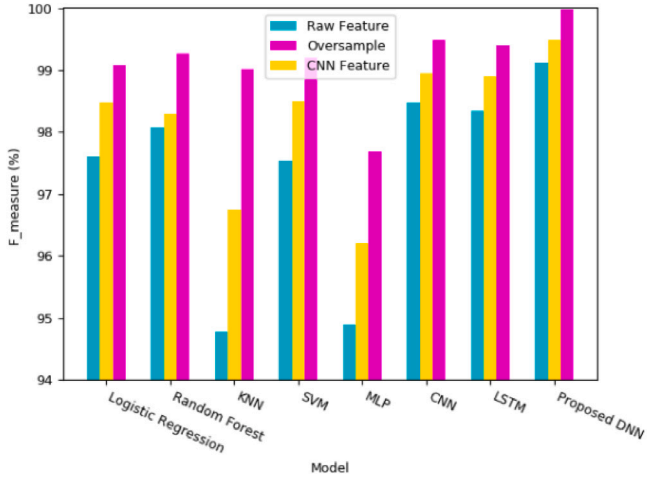

Fig. 20. F1 score of models on IoT dataset.

of all models improved after over-sampling, and our proposed deep network performed better than the other classifiers. This shows that the generated data follows the original data distribution well, and increasing the data helps the models more general and robust. Also, by utilizing the extracted feature using CNN, the performance of all models has enhanced. Generally, one advantage of deep networks is to extract a conceptual space from the data. In addition, the accuracy, F1, and other metrics of our proposed deep network are superior to almost all machine learning algorithms and also LSTM and CNN. As mentioned in the previous sections, our proposed deep network combines CNN and LSTM that considers both short and long dependencies concurrently. Also, selected optimal structure and fine-tuning of the network hyper-parameters after performing numerous experiments are other important keys.

For a more comprehensive evaluation of the proposed method and the effect of oversampling by BGAN, we have compared our work with a couple of new works, given in Tables 6–8. As can be seen, MalGan is more efficient than other methods. Therefore, even with a limited number of training samples, it can detect malware with reasonable accuracy. In [37] and [60], due to the network structure and input type, which is images, more samples are needed for training the network attributes. [35] and [19] have also performed poorly due to their feature engineering phase with limited data. Also, in [30] despite our method, only short-term dependencies between features are considered. In our method, because of selecting the optimal network structure and using raw bytecode as a feature, overhead and complexity is very trivial compared to other methods except Namavar et al. [30], which required time for extracting feature is not considered. Therefore, our proposed method can be used as a real-time malware detection system.

7. Conclusion and future research

In this paper, a new method for detection and generation of new IoT-edge malware samples based on raw bytes of the header has been presented. Also, an automated representation learning model for extracting a conceptual space of the raw data is implemented, which improves the training process of boundary-seeking GAN and enhances the accuracy of classifiers. Due to some limitations of GAN, boundary-seeking GAN has been used to generate new malware sample signatures using limited training data. The experimental results show that the accuracy of some classifiers have improved by about 4%, which shows the impact of generated data on the generalization and robustness of the classifiers. We also used a deep neural network, which simultaneously captures local and global dependencies on data to detect malware samples. The experimental results show that the proposed deep model outperforms CNN and LSTM models in terms of accuracy. Also, employing the attention mechanism to enhance the impact of relevant features in the classification increased the model's accuracy. Since our approach held a reliable performance even with limited training data, it can be used in settings with insufficient or

imbalanced data. Moreover, our method manages sequential data and can be employed to detect malware based on network traffic, opcode, or other sequential data.

In this paper, a GAN model has been used to generalized the model and oversampled the dataset. However, for future work, we intend to utilize GAN to generate and detect adversarial examples. Adversarial examples are among the most critical attacks that threaten machine learning models. Besides, in this paper, we generated signatures of malware samples that were not executable and accordingly used statistical methods to evaluate them. In the future, we intend to generate malware samples and analyze them using other methods such as executing in a virtual environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM Comput. Surv. 44 (2) (2008) 1–42.

[2] C. Beek, C. Castillo, C. Cochin, A. Dolezal, Mcafee Threats Report: December 2018, McAfee Labs, New York, NY, USA, 2018, pp. 3–7.

[3] J. Sakhnini, H. Karimipour, A. Dehghantanha, R.M. Parizi, G. Srivastava, Security aspects of internet of things aided smart grids: A bibliometric survey, Internet Things (2019) 100111.

[4] H. HaddadPajouh, A. Dehghantanha, R.M. Parizi, M. Aledhari, H. Karimipour, A survey on internet of things security: Requirements, challenges, and solutions, Internet Things (2019) 100129.

[5] A. Al-Abassi, H. Karimipour, A. Dehghantanha, R.M. Parizi, An ensemble deep learning-based cyber-attack detection in industrial control system, IEEE Access 8 (2020) 83965–83973.

[6] J.-Y. Kim, S.-J. Bu, S.-B. Cho, Malware detection using deep transferred generative adversarial networks, 2017, pp. 556–564.

[7] J.O. Kephart, Automatic extraction of computer virus signatures, 1994, pp. 178–184.

[8] E. Gandotra, D. Bansal, S. Sofat, Malware analysis and classification: A survey, J. Inf. Secur. 2014 (2014).

[9] M.D. Preda, M. Christodorescu, S. Jha, S. Debray, A semantics-based approach to malware detection, ACM SIGPLAN Not. 42 (1) (2007) 377–388.

[10] N. Milosevic, A. Dehghantanha, K.-K.R. Choo, Machine learning aided android malware classification, Comput. Electr. Eng. 61 (2017) 266–274.

[11] H. HaddadPajouh, A. Dehghantanha, R. Khayami, K.-K.R. Choo, A deep recurrent neural network based approach for internet of things malware threat hunting, Future Gener. Comput. Syst. 85 (2018) 88–96.

[12] A. Azmoodeh, A. Dehghantanha, K.-K.R. Choo, Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning, IEEE Trans. Sustain. Comput. 4 (1) (2018) 88–95.

[13] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, K.-K.R. Choo, An opcode-based technique for polymorphic internet of things malware detection, Concurr. Comput.: Pract. Exper. 32 (6) (2020) e5173.

[14] S.M. Tahsien, H. Karimipour, P. Spachos, Machine learning based solutions for security of internet of things (iot): A survey, J. Netw. Comput. Appl. 161 (2020) 102630.

[15] N. Peiravian, X. Zhu, Machine learning for android malware detection using permission and api calls, 2013, pp. 300–305.

[16] T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, A multimodal deep learning method for android malware detection using various features, IEEE Trans. Inf. Forensics Secur. 14 (3) (2018) 773–788.

[17] H. Hashemi, A. Hamzeh, Visual malware detection using local malicious pattern, J. Comput Virol. Hacking Tech. 15 (1) (2019) 1–14.

[18] C.H. Kim, E.K. Kabanga, S.-J. Kang, Classifying malware using convolutional gated neural network, 2018, pp. 40–44.

[19] M. Farrokhmanesh, A. Hamzeh, Music classification as a new approach for malware detection, J. Comput. Virol. Hacking Tech. 15 (2) (2019) 77–96.

[20] J.-Y. Kim, S.-J. Bu, S.-B. Cho, Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders, Inform. Sci. 460 (2018) 83–102.

[21] R.D. Hjelm, A.P. Jacob, T. Che, A. Trischler, K. Cho, Y. Bengio, Boundary-seeking generative adversarial networks, 2017, arXiv preprint arXiv:1702.08431.

[22] A.N. Jahromi, J. Sakhnini, H. Karimpour, A. Dehghantanha, A deep unsupervised representation learning approach for effective cyber–physical attack detection and identification on highly imbalanced data, in: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, 2019, pp. 14–23.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, 2014, pp. 2672–2680.

[24] C. Yinka-Banjo, O.-A. Ugot, A review of generative adversarial networks and its application in cybersecurity, Artif. Intell. Rev. (2019) 1–16.

[25] H. Li, S. Zhou, W. Yuan, J. Li, H. Leung, Adversarial-example attacks toward android malware detection system, IEEE Syst. J. 14 (1) (2019) 653–656.

[26] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, J. Chen, Adversarial examples detection for xss attacks based on generative adversarial networks, IEEE Access 8 (2020) 10989–10996.

[27] D. Li, Q. Li, Adversarial deep ensemble: Evasion attacks and defenses for malware detection, IEEE Trans. Inf. Forensics Secur. 15 (2020) 3886–3900.

[28] H. Karimipour, A. Dehghantanha, R.M. Parizi, K.-K.R. Choo, H. Leung, A deep and scalable unsupervised machine learning system for cyber-attack detection in large-scale smart grids, IEEE Access 7 (2019) 80778–80788.

[29] Z. Moti, S. Hashemi, A. Namavar, Discovering future malware variants by generating new malware samples using generative adversarial network, 2019, pp. 319–324.

[30] A.N. Jahromi, S. Hashemi, A. Dehghantanha, K.-K.R. Choo, H. Karimipour, D.E. Newton, R.M. Parizi, An improved two-hidden-layer extreme learning machine for malware hunting, Comput. Secur. 89 (2020) 101655.

[31] M. Amin, B. Shah, A. Sharif, T. Ali, K.-l. Kim, S. Anwar, Android malware detection through generative adversarial networks, Trans. Emerg. Telecommun. Technol. (2019) e3675.

[32] Z. Xu, S. Ray, P. Subramanyan, S. Malik, Malware detection using machine learning based analysis of virtual memory access patterns, 2017, pp. 169–174.

[33] Z. Bazrafshan, H. Hashemi, S.M.H. Fard, A. Hamzeh, A survey on heuristic malware detection techniques, 2013, pp. 113–120.

[34] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, Novel feature extraction, selection and fusion for effective malware family classification, 2016, pp. 183–194.

[35] A. Kumar, K. Kuppusamy, G. Aghila, A learning model to detect maliciousness of portable executable using integrated feature set, J. King Saud Univ. Comput. Inf. Sci. 31 (2) (2019) 252–265.

[36] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, J. Chen, Detection of malicious code variants based on deep learning, IEEE Trans. Ind. Inf. 14 (7) (2018) 3187–3196.

[37] D. Gibert, C. Mateu, J. Planes, R. Vicens, Using convolutional neural networks for classification of malware represented as images, J. Comput. Virol. Hacking Tech. 15 (1) (2019) 15–28.

[38] E. Raff, J. Sylvester, C. Nicholas, Learning the pe header, malware detection with minimal domain knowledge, 2017, pp. 121–132.

[39] I. Goodfellow, Nips 2016 tutorial: Generative adversarial networks, 2016, arXiv preprint arXiv:1701.00160.

[40] B. Fuglede, F. Topsoe, Jensen-shannon divergence and hilbert space embedding, in: International Symposium OnInformation Theory, 2004. ISIT 2004. Proceedings. 31, IEEE, 2004.

[41] M. Arjovsky, L. Bottou, Towards principled methods for training generative adversarial networks, 2017, arXiv preprint arXiv:1701.04862.

[42] S. Martin Arjovsky, L. Bottou, Wasserstein generative adversarial networks, 2017.

[43] V. Nagarajan, J.Z. Kolter, Gradient descent gan optimization is locally stable, 2017, pp. 5585–5595.

[44] S. Nowozin, B. Cseke, R. Tomioka, F-gan: Training generative neural samplers using variational divergence minimization, in: Advances in Neural Information Processing Systems, 2016, pp. 271–279.

[45] F. Liese, I. Vajda, On divergences and informations in statistics and information theory, IEEE Trans. Inform. Theory 52 (10) (2006) 4394–4412.

[46] T. Rezaei, A. Hamze, An efficient approach for malware detection using pe header specifications, in: 2020 6th International Conference on Web Research (ICWR), IEEE, 2020, pp. 234–239.

[47] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C. Nicholas, Malware detection by eating a whole exe, 2017, arXiv preprint arXiv:1710.09435.

[48] T. Committee, et al., Tool interface standard (tis) executable and linking format (elf) specification version 1.2, 1995.

[49] L. Liu, B. Wang, Automatic malware detection using deep learning based on static analysis, in: International Conference of Pioneering Computer Scientists, Engineers and Educators, Springer, 2017, pp. 500–507.

[50] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, D.J. Inman, 1d convolutional neural networks and applications: A survey, 2019, arXiv preprint arXiv:1905.03554.

[51] F.A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with lstm, 1999.

[52] T. Liu, J. Bao, J. Wang, Y. Zhang, A hybrid cnn–lstm algorithm for online defect recognition of co2 welding, Sensors 18 (12) (2018) 4369.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[54] E. Barshan, A. Ghodsi, Z. Azimifar, M.Z. Jahromi, Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds, Pattern Recognit. 44 (7) (2011) 1357–1371.

[55] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, J. Mach. Learn. Res. 13 (1) (2012) 723–773.

[56] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357.
[57] P. Orponen, et al., Computational complexity of neural networks: a survey, Nordic J. Comput. (1994).
[58] K. He, J. Sun, Convolutional neural networks at constrained time cost, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 5353–5360.
[59] S. Hochreiter, Ja1 4 rgen schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997).
[60] Q. Le, O. Boydell, B. Mac Namee, M. Scanlon, Deep learning at the shallow end: Malware classification for non-domain experts, Digit. Investig. 26 (2018) S118–S126.

Zahra Moti is with the Machine Learning Lab, Department of Computer Science, Engineering and Information Technology, Shiraz University, Iran. she graduated from Shiraz University in Computer Security Engineering with a Master of Science degree program. she has been researching in computer security and Machine Learning.

Sattar Hashemi received the Ph.D. degree in computer science from the Iran University of Science and Technology, Tehran, Iran, in conjunction with Monash University, Melbourne, Australia, in 2008. Following academic appointments with Shiraz University, Shiraz, Iran, where he is currently an Associate Professor with the Electrical and Computer Engineering School. His research interests include machine learning, data mining, social networks, data stream mining, game theory, and adversarial learning.

Hadis Karimipour received a Ph.D. degree in Electrical Engineering from the University of Alberta in 2016. She was recipient of the prestigious Queen Elizabeth II scholarship in 2014 and 2015. Before joining the University of Guelph, she was a postdoctoral fellow at the University of Calgary working on anomaly detection and security analysis of the smart power grids. She is currently the director of SCPS Lab and an Assistant Professor at the School of Engineering at the University of Guelph, Guelph, Ontario. Her research interests include power system analysis, cyber–physical modeling, cyber-security of the smart grids, and parallel and distributed computing. She is a member of the IEEE and IEEE Computer Society. She serves as the Chair of the IEEE Women in Engineering (WIE) and chapter chair of IEEE Information Theory in the Kitchener–Waterloo section.

Ali Dehghantanha is the director of the Cyber Science Lab at the University of Guelph, Ontario, Canada. His lab is focused on building AI-powered solutions to support cyber threat attribution, cyber threat hunting and digital forensics tasks in the Internet of Things (IoT), Industrial IoT, and Internet of Military of Things (IoMT) environments. Ali has served for more than a decade in a variety of industrial and academic positions with leading players in Cyber-Security and Artificial Intelligence. Prior to joining UofG, he has served as a Sr. Lecturer in the University of Sheffield, UK and as an EU Marie-Curie International Incoming Fellow at the University of Salford, UK. He has Ph.D. in Security in Computing and a number of professional certifications including CISSP and CISM.

Amir Namavar Jahromi has been working toward the Ph.D. degree in artificial intelligence with Shiraz University, Shiraz, Iran, since 2012. He also has a master's degree in information technology from the Amirkabir University of Technology (Tehran Polytechnic) of Tehran, Tehran, Iran, and a bachelor's degree in information technology. He is currently the In Charge of Machine Learning Laboratory (MLL), Shiraz University. His research interests are deep neural networks, extreme learning machine (ELM), and machine learning applications in computer security and computer networks.

Lida Abdi received the B.Sc. degree in computer engineering from Shiraz Payamnoor University in 2010, and the M.Sc. degree in artificial intelligence from Shiraz University, Iran, in 2013. Her research interests include machine learning, data mining, and class imbalance learning.

Fatemeh Alavi is a Ph.D. candidate of Artificial Intelligence at Shiraz University. she is with the Machine Learning Lab, Department of Computer Science, Engineering and Information Technology, Shiraz University, Iran.