IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 7, 2021, accepted March 27, 2021, date of publication April 8, 2021, date of current version April 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3071763

Intrusion Detection Based on Sequential Information Preserving Log Embedding Methods and Anomaly Detection Algorithms

CZANGYEOB KIM[1], MYEONGJUN JANG[2], SEUNGWAN SEO[1], KYEONGCHAN PARK[1], AND PILSUNG KANG[1]

[1]School of Industrial Management Engineering, Korea University, Seoul 02841, Republic of Korea
[2]Department of Computer Science, University of Oxford, Oxford OX1 3QD, U.K.

Corresponding author: Pilsung Kang (pilsung_kang@korea.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea Government (MSIT) under Grant NRF-2019R1F1A1060338 and Grant NRF-2019R1A4A1024732.

ABSTRACT Previous methods for system intrusion detection have mainly consisted of those based on pattern matching that employs prior knowledge extracted from experts' domain knowledge. However, pattern matching-based methods have a major drawback that it can be bypassed through various modified techniques. These advanced persistent threats cause limitation to the pattern matching-based detecting mechanism, because they are not only more sophisticated than usual threats but also specialized in the targeted attacking object. The defense mechanism should have to comprehend unusual phenomenons or behaviors to successfully handles the advanced threats. To achieve this, various security techniques based on machine learning have been developed recently. Among these, anomaly detection algorithms, which are trained in unsupervised fashion, are capable of reducing efforts of security experts and securing labeled dataset through post analysis. It is further possible to distinguish abnormal behaviors more precisely by training classification models if sufficient amounts of labeled dataset is obtained through post analysis of anomaly detection results. In this study, we proposed an end-to-end abnormal behavior detection method based on sequential information preserving log embedding algorithms and machine learning-based anomaly detection algorithms. Contrary to other machine learning based system anomaly detection models, which borrow domain experts' knowledge to extract significant features from the log data, raw log data are transformed into a fixed size of continuous vector regardless of their length, and these vectors are used to train the anomaly detection models. Experimental results based on a real system call trace dataset, our proposed log embedding method with unsupervised anomaly detection model yielded a favorable performance, at most 0.8708 in terms of AUROC, and it can be further improved up to 0.9745 with supervised classification algorithms if sufficient labeled attack log data become available.

INDEX TERMS System anomaly detection, cyber security, system log embedding, advanced persistent threat, ADFA-LD.

I. INTRODUCTION

The importance of potential threats of cyber attack has been constantly increasing since the exponential growth of computer network and application usage [1]. The intrusion detection system is a representative defense mechanism to protect a system from the continuously increasing attacks which are designed more precisely [2]. The intrusion detection system can be divided into two categories: pattern or signature-based systems and machine learning-based systems [3]. The pattern-based system gives an alarm to system administrators when a newly arrived log is matched with one of pre-obtained abnormal signatures [4]. This pattern-based detection system has some obvious limitations. First, attacks must be known through the analysis performed by security experts because the method heavily relies on predefined patterns. Second, proper customizing for individual companies or organizations is indispensable due to the difference in their computing system [5]. Third, pattern based methods cannot cope with all existing and newly created attacks that are growing exponentially. The intruder attempts a novel offense approach to bypass possible

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojie Ju.

detection patterns [6]. Therefore, the pattern-based defense mechanism can be easily incapacitated. To prevent this, not only a tremendous amounts of pre-defined patterns are necessary, but also they should be constantly and timely updated to successfully defend the system.

On the other hand, the machine learning-based intrusion detection methods can discover unknown novel attacks [4]. In spite of this advantage, researchers for machine learning-based intrusion detection methods have been struggling with obtaining a high-quality dataset for training and evaluating proposed detection algorithms [7]. The datasets used in previous researches were hard to release publicly because of privacy issues requiring anonymization. Although there are a few publicly available dataset for the research purpose, they do not reflect recent trends of intrusion and lack diversity of traffic and attack types since they were released even a few decades ago [2]. Additional practical limitation of the past studies on machine learning-based detection system is that they formulate the detection system as a classification problem under an unrealistic assumption that sufficient number of attack examples are available before training the model. However, this assumption is impractical since only a few abnormal behavior data exist among abundant normal behavior data in real industry. Therefore, an effective approach that can train the detection model without actual attack data is essentially required to apply machine learning-based intrusion detection system to the real industry. Another issue on machine learning-based intrusion detection models is that most machine learning algorithms take a fixed size of vector as an input data. However, system log data are generally variable length of tokens or commands that are neither a fixed length nor real valued. To handle this problem, most of previous works employed frequency based representation methods such as Bag-of-Words (BoW) and term frequency-inverse document frequency (TF-IDF). However, these approaches have a limitation that they cannot reflect sequential information of system log data.

In this study, we propose an effective intrusion detection model based on sequential information preserving log embedding methods and anomaly detection algorithms for Linux environment. In order to transform a system call trace with variable length to a fixed-dimensional real valued vector, we employed Doc2vec [8], recurrent neural network-based auto-encoder (RNN-AE), and recurrent neural network-based denosing auto-encoder (RNN-DAE) which can preserve the sequential information. To verify our detection model, we conducted an experiment based on the ADFA-LD dataset [9] that contains Linux system call trace. In addition, under the ideal assumption that we obtained sufficient amount of labeled data for every types of attacks, we trained supervised classification models and compared their performances with those of anomaly detection models to figure out how much the proposed anomaly detection models.

The rest of this paper is organized as follows. In Section 2, we briefly review the past researches on machine learning-based intrusion detection. In Section 3, we demonstrate long embedding methods that transform a variable length of Linux call trace to a fixed-dimensional real-valued vector and anomaly detection algorithms that we employed in this study. In Section 4, we illustrate the ADFA-LD dataset used to verify the proposed models. In section 5, experiment results for different embedding methods, anomaly detection and classification algorithms, and training/validation compositions are discussed. Finally, in Section 6, we conclude our present study which leads us to future research directions.

II. RELATED WORK

Machine learning-based intrusion detection studies can be be divided into two categories: supervised learning-based models and unsupervised learning-based models.

A. SUPERVISED LEARNING-BASED INTRUSION DETECTION

The previous supervised learning-based intrusion detection studies were performed mainly based on DARPA 1998, KDD 99, UNM, NSL-KDD, UNSW-NB15, and ADFA-LD datasets [10]–[14]. Chen *et al.* [15] attempted to detect intrusion by training supervised machine learning models on DARPA 1998 dataset. They used frequency-based methods and TF-IDF to vectorize data and trained support vector machine (SVM) and artificial neural networks (ANN). Zhang and Zulkernine [16] employed the Random Forest algorithm to detect intrusions for KDD 99 dataset. The Random Forest algorithm was trained in a supervised manner which requires a large amount of labeled normal and intrusion samples. Therefore, Zhang and Zulkernine [16] utilized protocol information which can be easily acquired by a simple processing technique as labels and trained the Random Forest model. Creech and Hu [17] applied the extreme learning machine (ELM), a new type of ANN, to KDD 99, UNM dataset and ADFA-LD dataset. Borisaniya and Patel [18] attempted to construct a binary classification model and a multi-class classification model for ADFA-LD and ADFA-WD dataset. They employed various n-gram features to represent a system call trace to a fixed sized vector and trained supervised classification models such as J48 and IBk. Haider *et al.* [19] extracted four statistics from the ADFA-LD dataset and classified them using k-NN and SVM to make real-time intrusion detection possible. Recently, deep learning-based intrusion detection has been actively studied. Al-Zewairi *et al.* [20] compared the performance of different activation functions with and without dropout technique using a feed-forward ANN structure. The rectifier function without the dropout yielded the best classification accuracy for the UNSW-NB15 dataset. Kwon *et al.* [21] employed the convolutional neural networks (CNN) as the backbone structure and compare three levels of model depth: shallow, moderate, and deep. They used the NSL-KDD dataset and found that detection performance does not increase in accordance with the model depth, and the performance of CNN structure is poorer than Seq2Seq-LSTM structure. Lin *et al.* [22] used the

LeNet-5 structure to classify attacks in the KDD 99 dataset. They performed feature reduction using information gain and then converted the original input into a 32×32 image to be fed to the CNN model. Kim *et al.* [23] proposed a new encoding schema that transforms the original feature vector into an RGB image and used GoogleNet to classify attacks. Anani and Samarabandu [24] conducted a comparative study of various recurrent neural network (RNN) structure such as LSTM, Bi-LSTM, Skip-LSTM, and GRU. Fu *et al.* [25] proposed a binary classification model that combines LSTM, mean pooling, and logistic regression to distinguish attack from normal instances. They evaluated the performance based on the NSL-KDD dataset and showed excellent classification performance in terms of detection rate and classification accuracy. Jiang *et al.* [26] proposed a way to train the model for each channel using the same structure proposed by Fu *et al.* [25] and to detect attacks in multi-channel by voting the model's predicted results.

B. UNSUPERVISED LEARNING BASED INTRUSION DETECTION RESEARCHES

There have been some attempts to combine unsupervised learning and supervised learning approaches. Zhang and Chen [27] proposed to extract significant features using RBM, an unsupervised learning method, based on which SVM and ANN are trained to classify normal and attack instances. Andresini *et al.* [28] proposed a method of training an autoencoder to reconstruct the training data and then using this reconstructed samples to train a new autoencoder for anomaly detection or to use it for classification model training. Andresini *et al.* [29] also proposed a model that trains the autoencoder for normal samples and the autoencoder for attack samples separately. Then, the original input and the output of both autoencoders are used to train a CNN classifier.

There are also several studies that employed purely unsupervised machine learning algorithms, i.e., anomaly detection algorithms, for intrusion detection. Heller *et al.* [30] detected malicious programs through one-class SVM (1-SVM). They used Microsoft Windows registry information as a training dataset under the assumption that programs primarily used by individual users are consistent. The most similar research that shares several common attributes with our study is that of Xie *et al.* [12] which applied the 1-SVM to the ADFA-LD dataset. Xie *et al.* [12] recorded the change of performance regarding to hyperparameters of the model for different length of n-grams. However, there is a limitation that it cannot represent sequential information and latent meaning of sequence because the length of n-grams was fixed and too short. Kabir and Luo [31] considered unsupervised algorithms such as K-Means clustering (KMC), Self-Organizing Map (SOM), Deep Autoencoder Gaussian Mixture Model, and adversarially learned anomaly detection (ALAD) for intrusion detection. Although algorithms achieved the best performance differed depending on the dataset, KMC and SOM resulted in a stable performance while ALAD proved to detect minority attacks well. There are only a few researches about intrusion detection based on unsupervised learning compared to that of supervised learning-based researches. The common issue arise from those researches were the practical applicability of the developed model to real systems due to its high false alarm rate. In addition, previous works were not able to reflect sequential information of system call trace since they used frequency based counting method to transform a system call trace to a numeric vector. To alleviate these problems, we employed distributed representations methods and RNN-AE to transform a variable length of character-based sequential system call trace to a fixed size of numeric vectors to preserve the sequential information and make it easy to train machine learning algorithms that require a fixed size of numeric vectors as an input data.

III. METHODS

The overall framework of the proposed intrusion detection system is illustrated in Figure 1. We used Linux system call traces as a source of system intrusion detection. In the first phase, these system call traces are transformed into a fixed size of numeric vectors based on three distributed representation models: Doc2Vec, RNN-AE, and RNN-DAE. In the second phase, three machine learning-based anomaly detection models, i.e., isolation forest (IF), local outlier factor (LOF), and 1-SVM are trained based on normal system call trace only and intrusion detection performance are compared based on other normal system call traces and actual intrusion trials. To understand the performance of anomaly detection models, Random Forests which is a binary classification model and is trained under an unrealistic assumption that sufficient intrusion samples are available during the training phase is also compared.

A. VECTORIZING SYSTEM CALL TRACE

1) Doc2Vec

Doc2vec is a popular distributed representation learning method originated from natural language processing (NLP) [32]. The purpose of Doc2Vec is to learn an appropriate numeric vector of a given document using the word vectors inside the document. Doc2vec learns the document vector by maximizing the probability to predict words that appear in the document. There are two approaches to learn the document vector: distributed memory (DM) and distributed bag of words (DBOW). The DM takes the document vector and k sequential word vectors as inputs and predicts the next word vectors. The k is known as window size and is a hyper-parameter of the algorithm. On the other hand, the DBOW takes only the document vector as an input and predicts randomly sampled words that appear in the document. The structures of DM and DBOW are shown in Figure 2. Although Doc2Vec was proposed for NLP, it can be applied to any type of dataset in which a set (document) consists of a sequence of items (words). In this study, each system call trace is considered as a document whereas each system call is considered as a word. Doc2vec embedding

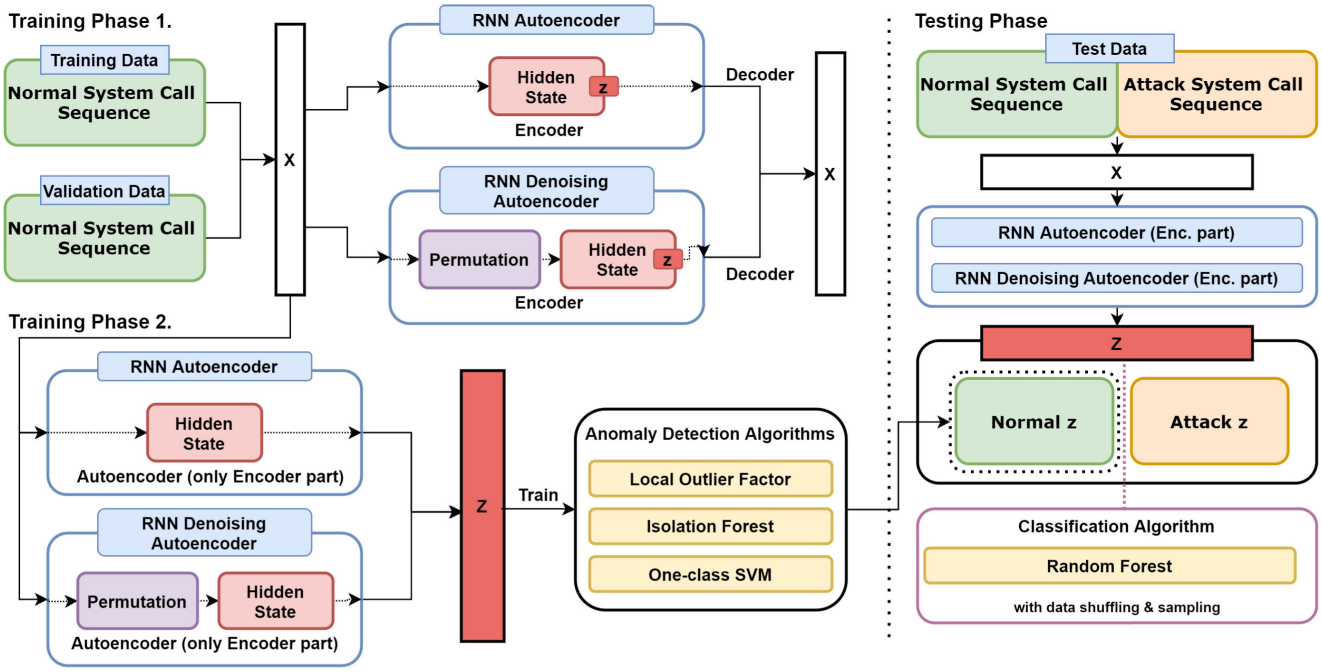


FIGURE 1. Overall framework for an intrusion detection: Model training is divided into two phases. In the first phase, we trained RNN-AE whose objective function is to reconstruct the normal system call sequence as much as possible. In the second phase, the encoder part of the RNN-AE is only used to obtain the hidden representation z for the normal system call sequences. Then, an anomaly detection model is trained based on these hidden representations z of normal system call sequences. In the test phase, an input is converted into a hidden representation vector by using the encoder part of the RNN-AE and is transferred to the anomaly detection and classification algorithms, respectively.

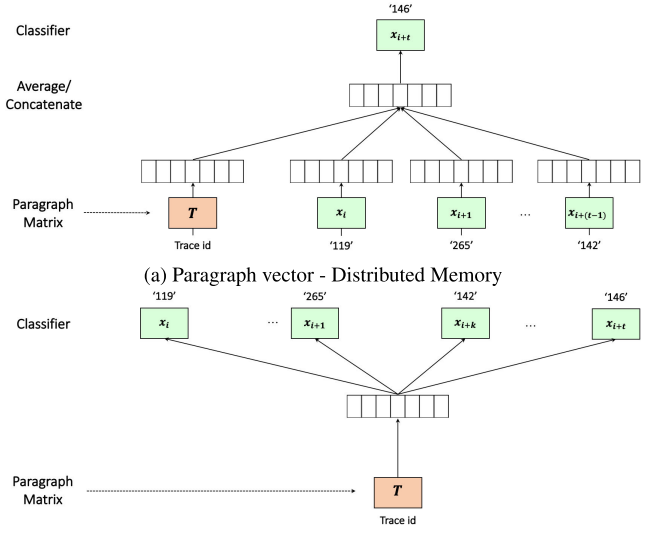


FIGURE 2. Structure of Doc2vec models: We assume a series of system call sequences as a single document and use Doc2vec model to obtain a fixed-length vector representing each sequence. Among the two possible models, we employed Distributed Memory structure.

method cannot preserve sequential information and is used to compare the performance difference with our proposed embedding method.

2) RNN-BASED AUTO-ENCODER (RNN-AE)

Auto-encoder in a neural network-based unsupervised learning method which encodes an input data to a lower-dimensional vector and is trained to reconstruct (decode) the input data as similar as possible [33]. The preceding part, compressing the input data to the lower-dimensional vector, is called the *encoder*, and the following part, reconstructing the output from the latent vector, is called the *decoder*. In this study, we construct Seq2seq model-based auto-encoder [34], [35] to process system call traces. The Seq2seq model employs RNN to construct the encoder and the decoder. The encoder compresses the information of input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ to the fixed vector $\mathbf{v}$:

$$\mathbf{h}_i = f(\mathbf{x}_i, \mathbf{h}_{i-1}), \tag{1}$$

$$\mathbf{v} = q(\{\mathbf{h}_1, \ldots, \mathbf{h}_T\}), \tag{2}$$

where $\mathbf{x}_i$ is the input vector of the i^{th} token, T is a total length of the input sequence, and $\mathbf{h}_i$ is the hidden state of the i^{th} sequence. f and q are nonlinear functions, where $q(\{\mathbf{h}_1, \ldots, \mathbf{h}_T\}) = \mathbf{h}_T$.

The decoder takes a vector $\mathbf{v}$ and previously predicted sequence $\mathbf{y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_{t'-1}\}$ as an input and is trained to maximize the probability of predicting the next token $\mathbf{y}_{t'}$. In other words, the learning object of the decoder is maximizing the probability of predicting target sequence $\mathbf{y}$ which is computed as follows:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \{\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}\}, v). \tag{3}$$

The conditional probability at time t is defined as follows:

$$p(\mathbf{y}_t|\{\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}\}, v) = g(\mathbf{y}_{t-1}, \mathbf{s}_t), \tag{4}$$

where g is a nonlinear function and $\mathbf{s}_t$ is a hidden state of the decoder at time t. The target $\mathbf{y}$ is identical to the input $\mathbf{x}$ because the objective of the auto-encoder is reconstructing the input itself. The objective function of the AE is defined as follows:

$$L(\theta_{enc}, \theta_{dec}) = E_{\tilde{x}\sim d(e(\mathbf{x}))}[\triangle(\hat{\mathbf{x}}, \mathbf{x})], \tag{5}$$

where e and d denotes encoder and decoder, respectively, and θ_{enc}, θ_{dec} are their parameters. In this study, we employed bi-directional RNN structure as shown in Figure 3.

3) DENOISING RNN-BASED AUTO-ENCODER (RNN-DAE)

The original auto-encoder has a limitation that it tends to overfit and is very sensitive to small input perturbations. To overcome this problem, denoising auto-encoder (DAE), which adds a little noise to the input data, has been proposed [36]. The learning objective of DAE is reconstructing original input data $\mathbf{x}$ from the noise added input data $\mathbf{C}(\mathbf{x})$ to make the auto-encoder more robust. where $\mathbf{C}$ is a corruption model which performs a role of adding noise. The objective function of the DAE is defined as follows:

$$L(\theta_{enc}, \theta_{dec}) = E_{\tilde{x}\sim d(e(\mathbf{C}(\mathbf{x})))}[\triangle(\hat{\mathbf{x}}, \mathbf{x})]). \tag{6}$$

We designed the same corruption model used by Lample *et al.* [37]. The corruption model adds noise to the input sequence in two different ways. One approach is eliminating tokens appearing in the sequence with the probability p_{wd}. The other approach is permutation that slightly shuffles the sequence order. In order to permute the order of the sequence, the random vector $\mathbf{q}$ is generated as follows:

$$q_i = i + \mathbf{U}(0, k + 1), \tag{7}$$

where $\mathbf{U}$ is an uniform distribution. Next, the input sequence is sorted based on the values of vector $\mathbf{q}$. If $k + 1 < 0$, the order of tokens does not change and the identical sequence is generated. On the other hand, if $k + 1 = +\infty$, the order of all tokens might be changed. In this study, we set the value of $p_{wd} = 0.1$ and $k = 3$ as used in Lample *et al.* [37].

B. ANOMALY DETECTION ALGORITHMS

1) 1-SVM

The 1-SVM is a model-based anomaly detection algorithm which trains its decision hyperplane to locate normal data as far as possible from the origin. The objective function of 1-SVM is as follows:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{vn}\sum_{i=1}^{n}\epsilon_i - \rho, \tag{8}$$

$$s.t \quad \mathbf{w}\cdot\Phi(\mathbf{x}_i) \geq \rho - \epsilon_i, \tag{9}$$

$$\epsilon_i \geq 0 \quad for\ i^{\forall}, \tag{10}$$

where $\mathbf{x}_i$ is a data object, $\mathbf{w}$ is the trainable parameter that determines the hyperplane, and Φ is a nonlinear function. v is a hyper-parameter that controls the ratio of support vectors that are either on the hyperplane or between the origin and the hyperplane. ϵ_i is the penalty term given to the observations between the origin and the hyperplane. The above equation can be transformed to following dual optimization problem:

$$\min \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}, \alpha_i\alpha_j K_\Phi(\mathbf{x}_i, \mathbf{x}_j) \tag{11}$$

$$s.t \quad \sum_{i=1}^{l}\alpha_i = 1, \tag{12}$$

$$0 \leq \alpha_i \leq \frac{1}{vl} \quad for\ i^{\forall}, \tag{13}$$

where α_i is the Lagrangian multiplier and K_Φ is a kernel function that transfers an input data to a higher dimensional feature space. There are many kernel functions such as linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. In this study, we used RBF kernel since it can generate most flexible decision boundary and shows the best performance in many real world applications [38]–[40]. Once the parameters of 1-SVM is optimized, the anomaly score can be obtained by following decision function:

$$f(\mathbf{x}_i) = sign(\mathbf{w}\cdot\Phi(\mathbf{x}_i) - \rho) = sign(\sum_{k=1}^{n} K_\Phi(\mathbf{x}_k, \mathbf{x}_i) - \rho). \tag{14}$$

2) ISOLATION FOREST (IF)

The IF is a decision tree-based anomaly detection algorithm. The main assumption of IF is that normal objects are densely located whereas abnormal objects are sparsely located far from normal objects so that it is much easier to isolate an abnormal object than normal objects by decision tree based on random split. Contrary to classification or regression tree which use the information gain to determine the best split criterion, IF used a randomly selected variable with randomly chosen value to split the dataset until the target object is isolated. In Figure 4 (a), isolating x_i, which is a normal object, generally requires a number of splits since many objects are located to the object. However, in Figure 4 (b), an abnormal object x_o can be isolated with only a few splits. Therefore, it is possible to detect anomalies by utilizing the number of split to isolate each object as anomaly score [41]. Since each tree is constructed based on random split, a sufficient number of trees are repeatedly constructed to isolate the target object and the average number of split is used to compute the anomaly score.

3) LOCAL OUTLIER FACTOR (LOF)

LOF is a density based anomaly detection algorithm that utilizes both local density information and distance information to ascertain abnormal observations [42]. The first step of LOF is defining k-distance of an observation $\mathbf{p}$ as follows:

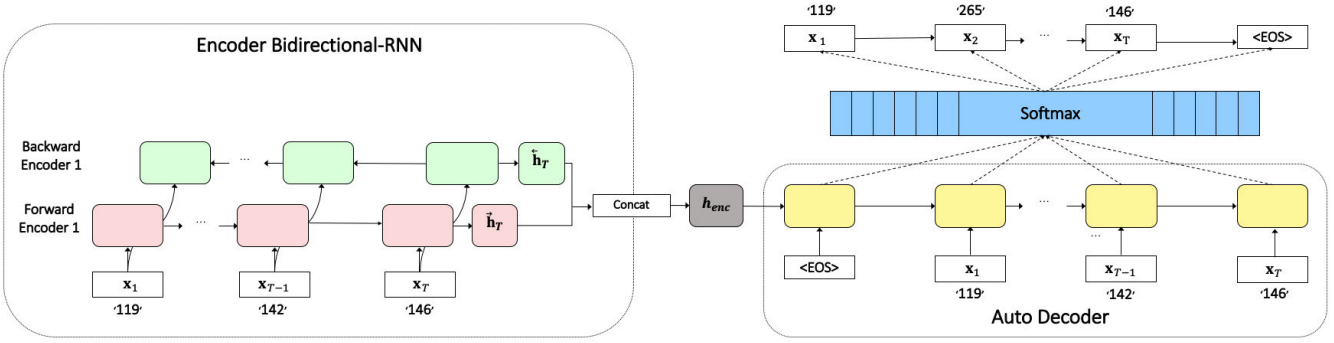


FIGURE 3. Bi-RNN-AE: We used Bi-RNN-AE to obtain a fixed-length vector preserving the sequential information of system call sequence. The Bi-RNN-AE is trained based only on normal system call sequences in the training dataset. Once the training of Bi-RNN-AE is completed, the fixed-length of hidden representations of normal system call sequences are used to train anomaly detection models.

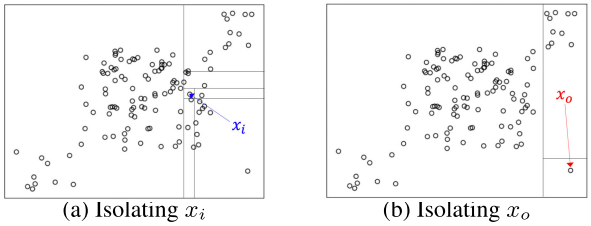
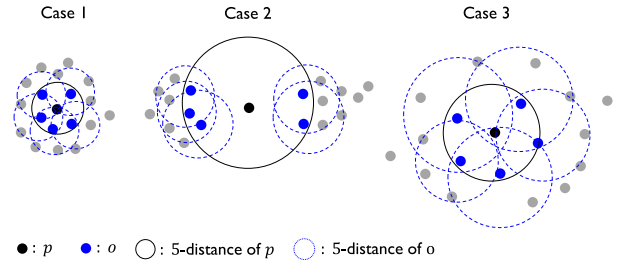


FIGURE 4. Examples of IF: isolating an abnormal object (right) generally requires fewer splits than isolating a normal object (left).



Case	$lrd_5(p)$	$lrd_5(o)$	$LOF_5(p)$
Case 1	Large	Large	Small
Case 2	Small	Large	Large
Case 3	Small	Small	Small

FIGURE 5. LOF examples with different *lrd* of target object and its neighbor objects: In case 2, target object p has a small lrd(p) value in the sparse area, and neighboring objects have a large lrd(o) value in the dense area. Because the LOF score is inversely proportional to lrd(p) and proportional to lrd(o), it is large for Case 2.

Definition 1: k-distance of an observation $\mathbf{p}$

For any positive integer k, the k-distance of an observation $\mathbf{p}$, denoted as k-distance($\mathbf{p}$), is defined as the distance $d(\mathbf{p}, \mathbf{o})$ between $\mathbf{p}$ and an observation $\mathbf{o} \in$ D such that:

1) for at least k observations $\mathbf{o}' \in D\backslash\{\mathbf{p}\}$ it holds that $d(\mathbf{p}, \mathbf{o}') \leq d(\mathbf{p}, \mathbf{o})$,
2) at least $k-1$ observations $\mathbf{o}' \in D\backslash\{\mathbf{p}\}$ it holds that $d(\mathbf{p}, \mathbf{o}') < d(\mathbf{p}, \mathbf{o})$.

The k-distance($\mathbf{p}$) is the distance to the k^{th} nearest neighbor observation considering ties. Next, the k-distance neighbor of an observation $\mathbf{p}$ is defined as follows:

Definition 2: k-distance neighborhood of an observation $\mathbf{p}$

Given the k-distance of $\mathbf{p}$, the k-distance neighborhood of $\mathbf{p}$, denoted as $N_k(\mathbf{p})$, contains every observation whose distance from $\mathbf{p}$ is not greater than the k-distance,

$$N_k(\mathbf{p}) = \{\mathbf{q} \in D\backslash\{\mathbf{p}\} | d(\mathbf{p}, \mathbf{q}) \leq k\text{-distance}(\mathbf{p})\}. \quad (15)$$

These observations $\mathbf{q}$ are called the k-distance neighborhood of $\mathbf{p}$.

The value k is a user-specified hyper-parameter. With k-distance($\mathbf{p}$) and k-distance neighbor of an observation $\mathbf{p}$, the reachability distance, which is the larger value between the observation $d(\mathbf{p}, \mathbf{o})$ and the k-distance($\mathbf{p}$), is defined as follows:

Definition 3: Reachability distance of an observation $\mathbf{o}$ from $\mathbf{p}$

$$\text{reachability-distance}_k(p, o) = max(k\text{-distance}(\mathbf{p}), d(\mathbf{p}, \mathbf{o})). \quad (16)$$

Based on the reachability distance, the local reachability density of an observation $\mathbf{p}$ is computed as follows:

Definition 4: Local reachability density of an observation $\mathbf{p}$

$$lrd_k(\mathbf{p}) = \frac{|N_k(\mathbf{p})|}{\sum_{\mathbf{o} \in N_k(\mathbf{p})} \text{reachability-distance}_k(\mathbf{p}, \mathbf{o})}. \quad (17)$$

$lrd_k(\mathbf{p})$ computes the local density around the observation $\mathbf{p}$. If $\mathbf{p}$ is in the middle of a dense area, reachability-distance$_k(\mathbf{p}, \mathbf{o})$ becomes small, resulting in a large value of $lrd_k(\mathbf{p})$. On the other hand, if $\mathbf{p}$ is in a sparse area, reachability-distance$_k(\mathbf{p}, \mathbf{o})$ becomes large, resulting in a small value of $lrd_k(\mathbf{p})$.

Finally, the LOF of an observation $\mathbf{p}$, which performs as an anomaly score, is computed as follows:

Definition 5: LOF of an observation $\mathbf{p}$

$$LOF_k(\mathbf{p}) = \frac{\frac{1}{lrd_k(\mathbf{p})} \sum_{\mathbf{o} \in N_k(\mathbf{p})} lrd_k(\mathbf{o})}{|N_k(\mathbf{p})|}. \quad (18)$$

The LOF score of objects $\mathbf{p}$ is inversely proportional to the local reachability density of $\mathbf{p}$ and is proportional to the

local reachability density of the objects belonging to $N_k(\mathbf{p})$. Therefore, observations having large LOF score are classified as the novelty. Figure 5 shows the LOF scores of three different cases. The $\mathbf{p}$ in Case 1 and Case 3 are located inside a dense (Case 1) or sparse (Case 3) area. Although the relative density around the $\mathbf{p}$ is different, but they are surrounded by the neighbors so that the anomaly scores of both cases become small. On the other hand, in Case 2, $\mathbf{p}$ is located in a sparse area between two small dense areas, its anomaly score becomes large.

C. PERFORMANCE MEASURES

To evaluate the performance of anomaly detection models, we used three evaluation metrics: area under receiver operating characteristic curve (AUROC), equal error rate (EER) and hit rate. We also used receiver operating characteristic (ROC) curve to understand the detection performance of each model. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. Therefore, it shows whether detection ratio and false alarm increase or decrease according to the threshold values [15]. The AUROC is an area under the ROC curve. The larger value of AUROC indicates the machine learning model performs better. The EER is the point where the false rejection rate (FRR) and the false acceptance rate (FAR) are equal. The lower the EER value, the higher the accuracy of the machine learning algorithm. The hit rate is a ratio of objects correctly predicted as an attack to the actual attacks. To evaluate the anomaly detection performance on observations having high anomaly scores, we defined the *k%-Hit rate*. It is calculated as the number of objects belong to the upper k% anomaly score and accurately predicted as an attack divided by the number of actual attacks. Therefore, the *k%-Hit rate* of random guessing model should be $\frac{k}{100}$. This value can serve as a reference point for evaluating the performance of anomaly detection model.

IV. EXPERIMENTS

A. DATASET

Most previous studies on system anomaly detection were conducted based on KDD 99 dataset. However, it has been almost 20 years since the introduction of KDD 99 dataset, the dataset cannot reflect the recent trends of system intrusion strategies. To overcome the limitation of KDD 99 dataset, the ADFA-LD dataset was recently released to reflect recent version of operating system and intrusion techniques. The ADFA-LD dataset consists of normal and attack traces which are represented as the sequence of Linux system call. The number of normal and attack observations of the ADFA-LD dataset is provided in Table 1. The attack data is distinguished in six different categories: `Adduser`, `FTP-Hydra`, `SSH-Hydra`, `Meterpreter`, `Java-based Meterpreter`, and `Web Shell`.

The system traces of ADFA-LD dataset have variable length which means that each system call trace consists of different number of system calls as shown in Table 2. Therefore, it is required to transform a system trace to a fixed-size vector value. Contrary to the previous studies that employed n-gram to represent a system call trace as a vector, we employed three embedding methods: Doc2vec, RNN-AE, and RNN-DAE as demonstrated in Section 3.1. In the case of Doc2vec, all vectors of training/validation normal traces and attack traces are learned simultaneously. In the case of RNN-AE and RNN-DAE, we first trained the model with normal traces in the training dataset.

It should be worth noting that the training and validation datasets have to share the same data distribution. In other words, they should be generated from the same mechanism but have different observation values so that the validation dataset can be used to evaluate the generalization ability of machine learning models trained based on the training dataset. However, through the experiments, we found that this assumption is violated for ADFA-LD dataset. Therefore, the final performance is evaluated on the model which is trained on randomly sampled training dataset from the mixture of original training and validation dataset.

TABLE 1. Number of samples in the ADFA-LD dataset.

Attack	Normal	
746	Train	Validation
	833	4,373

TABLE 2. The number of total calls in a single system call trace in the ADFA-LD dataset. The length of the system call trace varies between 0 and 4,494.

Percentile	N. calls (min)	N. calls (max)	Range (max-min)
$0\% \sim 10\%$	0	114	114
$10\% \sim 20\%$	115	143	28
$20\% \sim 30\%$	144	181	37
$30\% \sim 40\%$	182	229	47
$40\% \sim 50\%$	230	317	87
$50\% \sim 60\%$	318	385	67
$60\% \sim 70\%$	386	543	157
$70\% \sim 80\%$	544	882	338
$80\% \sim 90\%$	883	1,471	588
$90\% \sim 100\%$	1,472	4,494	3,022

B. IMPLEMENTATION DETAILS

We implemented Doc2vec using gensim library and Random Forest using scikit-learn library. RNN-AE and RNN-DAE were implemented with Tensorflow v1.15. The experimental hardware environment is as follows:

- CPU: Intel (R) Core (TM) i7-7700K CPU 4.20GHz
- GPU: NVIDIA GTX 1080Ti 11GB RAM
- RAM:32G

TABLE 3. Average performance of the anomaly detection algorithms (IF, LOF, 1-SVM). Concerning the vector dimension, 150-dimensional representation vectors resulted in the best performance. IF yielded the best performance, followed by 1-SVM with a little margin among the anomaly detection algorithms. Based on this result, we used IF as the anomaly detection algorithm in subsequent experiments.

Attack	Vector dim.	IF		LOF		1-SVM	
		AUROC	EER	AUROC	EER	AUROC	EER
Adduser	100	0.7329	0.2961	0.5866	0.4111	0.7283	0.2976
	150	**0.7717**	**0.2628**	0.5830	0.4123	0.7474	0.2690
	200	0.7450	0.2856	0.5369	0.4292	0.7171	0.2965
Hydra-FTP	100	0.7278	0.3360	0.6153	0.3896	0.7199	0.3262
	150	0.7405	0.3181	0.6196	0.4003	0.7378	**0.3130**
	200	**0.7421**	0.3177	0.5862	0.4184	0.7274	0.3132
Hydra-SSH	100	0.7329	0.3198	0.6343	0.3917	0.7304	0.3135
	150	**0.7599**	**0.2923**	0.6369	0.3863	0.7510	0.2968
	200	0.7394	0.3156	0.5847	0.4241	0.7293	0.3166
Java-Meterpreter	100	0.7627	0.2916	0.5925	0.4081	0.7342	0.2927
	150	**0.7665**	**0.2758**	0.5904	0.4182	0.7370	0.2817
	200	0.7662	0.2768	0.5791	0.4057	0.7320	0.2910
Meterpreter	100	0.6937	0.3466	0.5690	0.4419	0.6977	0.3261
	150	**0.7500**	**0.2817**	0.5933	0.4205	0.7316	0.3002
	200	0.7360	0.3096	0.5526	0.4296	0.7097	0.3061
Web Shell	100	0.7640	0.2918	0.6317	0.3869	0.753	0.2669
	150	**0.8129**	0.2477	0.6475	0.3830	0.8020	**0.2401**
	200	0.7923	0.2681	0.6004	0.4058	0.7887	0.2589

C. EXPERIMENTAL RESULTS

We first evaluated the performance of unsupervised learning-based anomaly detection algorithms on various types of attacks. Next, under an ideal assumption that a sufficient amount of labeled data for each attack type, we trained a supervised classification model and compared its performance with that of anomaly detection algorithms.

1) PERFORMANCE COMPARISON OF ANOMALY DETECTION ALGORITHMS

We first compared the performance of three anomaly detection algorithms, i.e., IF, LOF, and 1-SVM, with different embedding vector sizes. We conducted the experiment by changing the vector embedding dimension to 100, 150, and 200. We recorded the average performance of the three embedding methods: Doc2vec, RNN-AE, and RNN-DAE for each anomaly detection model and embedding vector size. The anomaly detection algorithms were trained by using embedding vectors of normal training system call traces. The result of experiment is shown in Table 3. The best performance for each attack class is shown in bold with underline. The experimental result shows that IF algorithm trained with embedding vector size of 150 dimension yielded the best performance on average. For Hydra-FTP and Web Shell attacks, 1-SVM resulted in higher EER than IF. However, because the performance difference is marginal and1-SVM requires a separate hyperparameter tuning process, we discuss the following experimental results using the IF as the main anomaly detection model.

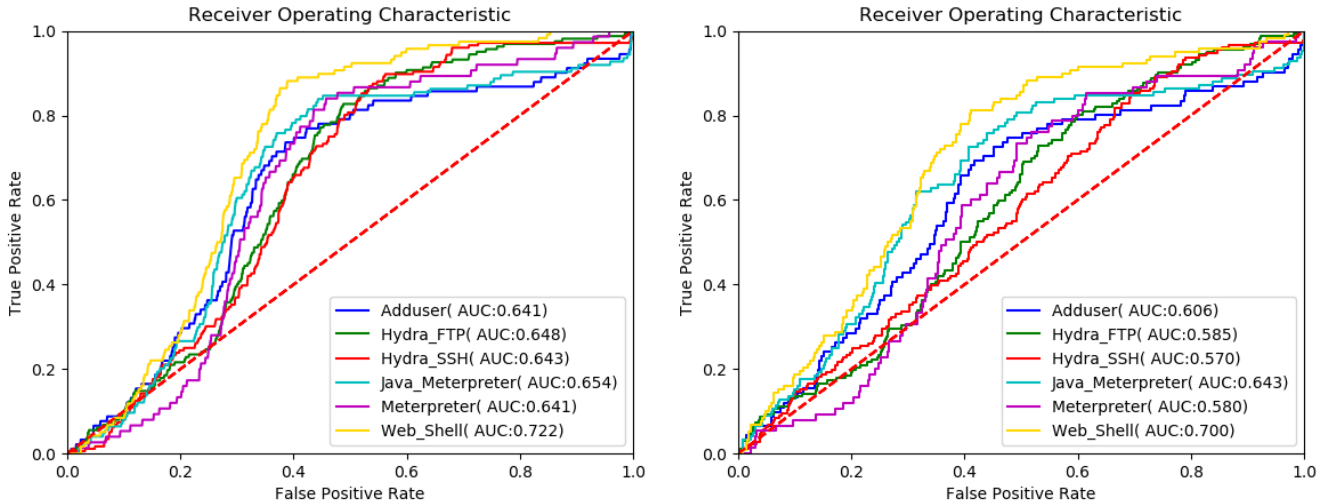
2) EXPERIMENTAL RESULT OF IF ON ORIGINAL NORMAL TRAINING AND VALIDATION DATASETS

We conducted more detailed experiments of IF trained with system call trace vectors of 150 dimension. The basic assumption of evaluating the performance of machine learning algorithms is that the data distributions of the training and validation dataset are identical. To check whether the ADFA-LD dataset satisfy this assumption, we conducted the following two experiments. In the first experiment, the normal system call traces in the training dataset were used to train the anomaly detection model and the normal system call traces in the validation dataset and the attack traces were used for evaluation as usually done in many studies. We conducted another experiment by changing the role of normal trances in the training and validation dataset, i.e., normal traces in the validation dataset was used to train the model, and the normal traces in the training dataset and the attack traces were used for evaluation. If the assumption on the identical data distribution is satisfied, the anomaly detection performances of two experiments should not be much different. In the following tables, we denoted the first and second case as 'Train' and 'Validation', respectively. We recorded the hit rate of 1%, 5%, 10% anomaly score, AUROC, and EER.

The anomaly detection performances with Doc2vec embedding vectors are shown in Table 4. The experimental result shows that 'Train' case and 'Validation' case yielded significantly different outcomes. In terms of hit rate, the 'Validation' case showed better detection performance. On the other hand, 'Train' case yielded higher AUROC and EER for all attack types. When looking at the ROC curve shown

TABLE 4. Anomaly detection performance of IF with Doc2Vec embedding. IF resulted in worse detection performances than random guess for most attack types in the training dataset, while its detection performances are slightly better than random guess in the validation dataset.

	Attack	Top 1% HR	Top 5% HR	Top 10% HR	AUROC	EER
IF + Doc2vec (Train)	Adduser	0.0110	0.0659	0.0879	0.6405	0.3407
	Hydra-FTP	0.0062	0.0559	0.0926	0.6476	0.3864
	Hydra-SSH	0.0057	0.0170	0.0795	0.6426	0.3853
	Java-Meterpreter	0.0000	0.0403	0.0806	0.6537	0.3306
	Meterpreter	0.0000	0.0267	0.0533	0.6408	0.3508
	Web Shell	0.0000	0.0424	0.0847	0.7222	0.3123
	Average	0.0038	0.0414	0.0798	0.6579	0.3510
IF + Doc2vec (Validation)	Adduser	0.0330	0.0659	0.1319	0.6062	0.3846
	Hydra-FTP	0.0247	0.0864	0.1296	0.5853	0.4411
	Hydra-SSH	0.0114	0.0625	0.1250	0.5696	0.4659
	Java-Meterpreter	0.0323	0.0806	0.1371	0.6429	0.3629
	Meterpreter	0.0000	0.0533	0.0800	0.5796	0.4111
	Web Shell	0.0085	0.0847	0.1610	0.6997	0.3305
	Average	0.0183	0.0722	0.1274	0.6138	0.3994



(a) using training dataset (b) using validation dataset

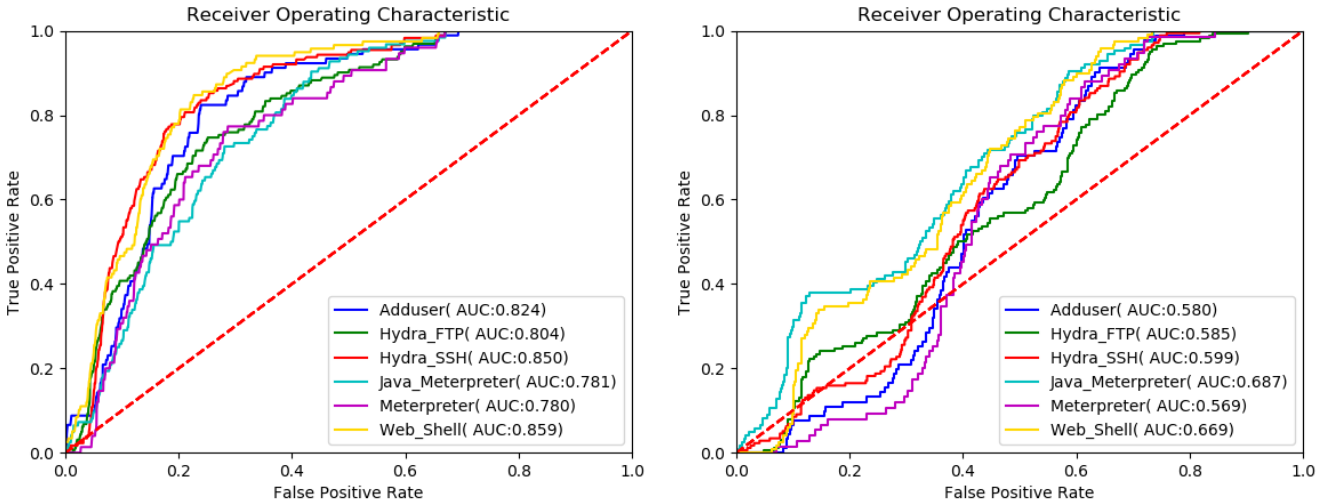
FIGURE 6. ROC curve of IF using Doc2vec embedding. The training dataset and validation dataset show noticeably different shapes.

in Figure 6, IF with 'Train' case and 'Validation' case shows noticeably different shapes: the TPR of IF with 'Train' increases rapidly beyond the FPR of 0.2 while the increasing rate is relatively moderate with 'Validation' case. However, the anomaly detection models of both cases performed just slightly better than random guess which leads us to the conclusion that Doc2vec embedding vectors are inappropriate to train the model for detecting unusual system call traces.

Table 5 shows the anomaly detection performance of IF with RNN-AE embedding vectors. With the RNN-AE embedding method, the detection performance gap between 'Train' and 'Validation' widens compared with the Doc2Vec embedding method. For the 'Train' case, IF can detect the unusual system call traces with around 0.8 AUROC score. Also, for observations which are in top 10% of abnormal scores, the performance improvement was 3 $\sim$ 4.5 times to that of random detection model. However, in the 'Validation' case, IF with RNN-AE embedding method yielded much lower detection performance than 'Train' case. As opposed to the 'Train' case, which achieved the average AUROC of around 0.8, the 'Validation' case resulted in the anomaly detection performance similar to that of the random guess model. Except for Java-Meterpreter attacks, the AUROC values for other attacks are between 0.5 and 0.6 (AUROC of random guess is 0.5) The ROC curve shown in Figure 7 also attest this tendency. In the 'Training' case, the curve significantly increases for all attack types, resulting in high AUROC scores. However, the anomaly detection model performed even worse than the random guess model with low false positive rate (when the cut-off is tight) for some attack types, and its increasing rate is not as sharp as that of 'Training' case.

TABLE 5. Anomaly detection performance of IF with RNN-AE embedding. Detection performances with RNN-AE embedding are better than Doc2vec embedding, but the performance differences between the training dataset and the validation dataset for the same types of attacks are quite different.

	Attack	Top 1% HR	Top 5% HR	Top 10% HR	AUROC	EER
IF + AE (Train)	Adduser	0.0659	0.0989	0.3187	0.8238	0.2340
	Hydra-FTP	0.0000	0.1790	0.3889	0.8041	0.2531
	Hydra-SSH	0.0170	0.1080	0.4659	0.8504	0.2045
	Java-Meterpreter	0.0242	0.0887	0.2581	0.7813	0.2806
	Meterpreter	0.0000	0.0400	0.2933	0.7797	0.2731
	Web Shell	0.0339	0.2119	0.4492	0.8587	0.2020
	Average	0.0235	0.1211	0.3624	0.8163	0.2412
IF + AE (Validation)	Adduser	0.0000	0.0000	0.0769	0.5804	0.4279
	Hydra-FTP	0.0000	0.0062	0.0988	0.5852	0.4483
	Hydra-SSH	0.0057	0.0284	0.0852	0.5990	0.4207
	Java-Meterpreter	0.0161	0.0806	0.1855	0.6873	0.3822
	Meterpreter	0.0000	0.0000	0.0267	0.5692	0.4267
	Web Shell	0.0000	0.0000	0.1019	0.6685	0.3966
	Average	0.0036	0.0192	0.0958	0.6149	0.4171



(a) Training dataset (b) Validation dataset

FIGURE 7. ROC curve of IF using RNN-AE embedding. The training dataset and validation dataset show noticeably different shapes.

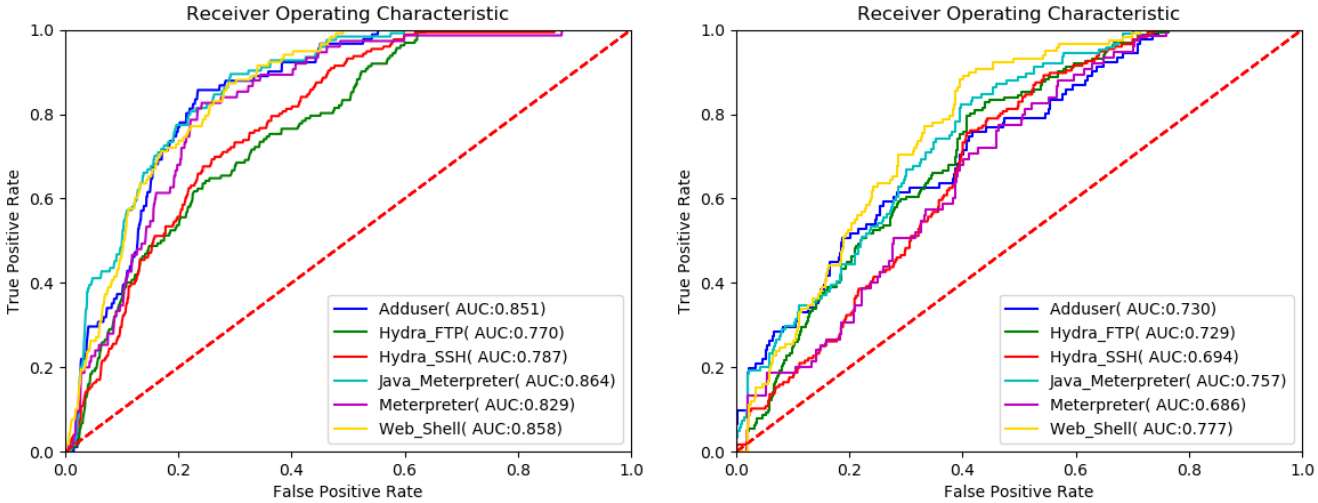
The anomaly detection performances of IF with RNN-DAE embedding vectors are shown in Table 6 and the corresponding ROC curves are shown in Figure 8. The result of RNN-DAE embedding vectors revealed the similar tendency with that of RNN-AE and Doc2Vec embedding vectors: the anomaly detection model can better detect anomaly objects when trained with the 'Train' dataset rather than 'Validation' dataset. In addition, the 'Train' case of the RNN-DAE embedding vectors showed the best performance among the three embedding methods. In terms of Top 5% HR, IF with RNN-DAE can detect intrusions five times more than random guessing, and on average, it has a performance of 0.8265 in terms of AUROC. Because all three embedding vectors resulted in significant performance difference between 'Train' and 'Validation' cases, we can conclude that the provided normal traces of training and validation dataset in ADFA-LD dataset have different data distributions. Among the three embedding methods, RNN-DAE yielded the highest AUROC and EER for most attack types in both cases. Hence, we employed RNN-DAE as a base system call trace embedding method for the following experiments.

3) EXPERIMENTAL RESULT OF IF ON RANDOMLY SHUFFLED NORMAL TRAINING AND VALIDATION DATASETS

Because the normal objects in the original training and validation datasets do not seem to be generated from the same distribution, we reorganize the datasets by shuffling them and randomly split them to the training and validation dataset and applied the IF with RNN-DAE embedding to investigate the generalized anomaly detection performance. In addition, to exploit the effect of the number of training objects on the detection performance, we varied the normal training objects

TABLE 6. Anomaly detection performance of IF with RNN-DAE embedding: RNN-DAE embedding outperforms Doc2vec and RNN-AE but the performance differences between the training dataset and the validation dataset are still significantly different.

	Attack	Top 1% HR	Top 5% HR	Top 10% HR	AUROC	EER
IForest + DAE (Train)	Adduser	0.0000	0.2967	0.3736	0.8508	0.2138
	Hydra-FTP	0.0062	0.1852	0.3333	0.7698	0.3148
	Hydra-SSH	0.0170	0.1534	0.2727	0.7867	0.2871
	Java-Meterpreter	0.0081	0.3871	0.4516	0.8644	0.2162
	Meterpreter	0.0133	0.2267	0.3333	0.8294	0.2211
	Web Shell	0.0508	0.2627	0.4237	0.8580	0.2288
	Average	0.0159	0.2520	0.3647	0.8265	0.2470
IForest + DAE (Validation)	Adduser	0.0989	0.1978	0.2857	0.7297	0.3626
	Hydra-FTP	0.0000	0.0802	0.2037	0.7287	0.3457
	Hydra-SSH	0.0170	0.1023	0.1648	0.6944	0.3762
	Java-Meterpreter	0.0484	0.1935	0.2661	0.7572	0.3226
	Meterpreter	0.0000	0.1333	0.1867	0.6855	0.3862
	Web Shell	0.0000	0.1525	0.2458	0.7768	0.2966
	Average	0.0273	0.1433	0.2255	0.7287	0.3483



(a) using training dataset (b) using validation dataset

FIGURE 8. ROC curve of IF using RNN-DAE embedding. The RNN-DAE embedding vectors yielded the best performance among the three embedding methods.

TABLE 7. AUROC of IF with RNN-DAE embedding based on randomly sampled training dataset.

Training dataset size	500	1000	1500	2000	2500	3000	3500	4000
Adduser	0.5021	0.6810	0.8184	0.8227	0.7712	0.8019	**0.8531**	0.7562
Hydra-FTP	0.6083	0.6765	**0.8241**	0.7608	0.7448	0.7601	0.7726	0.7653
Hydra-SSH	0.6002	0.6765	**0.8083**	0.7181	0.7058	0.6845	0.7379	0.7508
Java-Meterpreter	0.6604	0.7608	0.8766	0.8349	0.8198	0.8635	**0.8828**	0.7884
Meterpreter	0.4896	0.6344	0.7834	0.7812	0.7245	0.7774	**0.8306**	0.7149
Web Shell	0.6317	0.7089	0.8616	0.8286	0.8104	0.8088	**0.8708**	0.8130
Average	0.5821	0.6897	**0.8287**	0.7911	0.7628	0.7827	0.8246	0.7648

from 500 to 4,000 with the step size of 500. The experimental results under the aforementioned setting are shown in Table 7. When the number of normal training objects are insufficient, IF cannot detect the attacks irrespective of attack types. However, the detection performance improves in accordance with the number of normal training objects. Because Hydra-FTP and Hydra-SSH attacks have relatively simpler forms than other attacks, i.e., they are extremely fast

TABLE 8. AUROC of RF with randomly sampled RNN-DAE embeddings. We trained the RF with two training dataset sizes, i.e., 1,500 and 3,500, based on which IF with RNN-DAE achieved the best performances.

Training data size	Number of decision trees	Adduser	Hydra-FTP	Hydra-SSH	Java-Meterpreter	Meterpreter	Web Shell	Average
1500	10	0.8918	0.9614	0.9430	0.9148	0.9051	0.8636	0.9133
	50	0.9371	0.9730	0.9626	0.9525	0.9224	0.9137	0.9436
	100	0.9422	0.9725	0.9660	0.9588	0.9409	0.9206	0.9502
3500	10	0.8976	0.9527	0.9591	0.9209	0.9243	0.8726	0.9212
	50	0.9402	0.9795	**0.9745**	0.9625	0.9533	0.9308	0.9568
	100	**0.9538**	**0.9850**	**0.9745**	**0.9741**	**0.9661**	**0.9460**	**0.9666**

repetitive login attempts IF resulted in the highest performance with only 1,500 normal system call traces, whereas other attack types required 3,500 normal system call traces to achieve the best performance. It is also worth nothing that beyond some number, an increased number of normal objects does not help improve the detection performance anymore. In our experiment, anomaly detection algorithm trained with 4,000 normal system call traces resulted in lower AUROC than that with 3,500 normal system call traces.

4) EXPERIMENTAL RESULT ON RF ON RANDOMLY SHUFFLED NORMAL TRAINING AND VALIDATION DATASETS

In this study, we assume that it is very difficult and time consuming to obtain a sufficient number of labeled attack objects so that the intrusion detection model should be built based only on the unlabeled normal objects. One possible scenario is that once the anomaly detection-based intrusion detection model is running, the suspicious system call traces detected by the anomaly detection model can be reviewed by domain experts and can be determined whether they are actually intrusion trials or false alarms. Once these post-analyzed system call traces are accumulated, we can also employ a classification-based intrusion detection model by training it using the secured labeled data. Hence, we conducted additional experiment to see how much the detection performance can be improved when a sufficient attack objects can be available by employing Random Forest algorithm. To train the Random Forest model, we used the same RNN-DAE system call trace vectors which showed the best performance on the randomly sampled experiment. The only hyper-parameter of Random Forest is the number of individual trees so we selected the best number of trees among 10, 50, and 100 based on the 10 fold cross-validation result.

Table 8 shows that the intrusion detection performance can be improved above 0.95 in terms of AUROC if classification algorithms are trained based on sufficient attack objects. Once a sufficient number of trees are used for random sampling, all types of attacks are corrected detected: the lowest AUROC is 0.9460 (Web Shell) while the highest AUROC is 0.9850 (Hydra-SSH) Based on these results, we can design dual-phase models for intrusion detection system. In an early stage, an anomaly detection-based model is quickly trained based on the available normal objects. Then, when operating the anomaly detection-based models, domain experts participate to determine whether the alarmed objects are actually attack or not. As the attack objects are accumulated, a classification-based anomaly detection model is trained and the performances of the anomaly detection-based model and the classification-based anomaly detection model is compared. Once the classification-based model outperforms the anomaly detection-based model, the main intrusion detection model can be switched from the anomaly detection-based model to the classification-based model.

5) COMPARISON WITH PREVIOUS STUDIES

To validate the proposed method, we compared its detection performance with previously proposed other models. Some studies employed an anomaly detection approach [12], [13], [19] as we did in this study, whereas another study employed a supervised machine learning approach [18]. Since those studies employing anomaly detection approach did not explicitly provide the AUROC scores, we compared the hit rates of top 5% and 10%. Table 9 shows the hit rates of each model. Xie *et al.* [12], [13] first extracted a fixed-length vector using n-gram method. Then, anomaly detection algorithms such as k-NN, KMC, and 1-SVM, were applied. Haider *et al.* [19] extracted four statistical features and then attempted to classify them using k-NN and SVM. Because these embedded vectors do not preserve sequential information, it is not easy to fully preserve the semantic meaning of an entire system call trace. On the other hand, since our proposed RNN-DAE embedding vectors can preserve sequential information, much higher detection performances were obtained. In the case of supervised machine learning-based approach,

TABLE 9. Hit rates of anomaly detection models proposed in previous studies on the same dataset.

Algorithm	Top 5%	Top 10%
Frequency based features with k-NN, KMC [13]	≈ 0.1500	≈ 0.3050
Short sequence-based features with 1-SVM [12]	≈ 0.1800	≈ 0.3450
Four statistical features with k-NN [19]	≈ 0.1900	≈ 0.3600
RNN-DAE + IF (ours)	**0.2520**	**0.3647**

TABLE 10. AUROC of supervised machine learning algorithms.

Algorithm	AUC Score	Hyperparameter
SMO [18]	0.8610	Term-size=3
JRip [18]	0.9030	Term-size=1
Naïve Bayes [18]	0.9190	Term-size=3
J48 [18]	0.9270	Term-size=1
IBk [18]	**0.9680**	**k=3, Term-size=1**
RNN-DAE + RF (ours)	0.9667(Avg.)	Hidden size=150

there have been many papers describing AUC scores. Most notably, Borisaniya and Patel [18] conducted an experiment with several supervised learning algorithms, such as Naïve Bayes, SMO, LibSVM, IBk, kMeans, ZeroR, OneR, JRip, and J48. During the experiment, they changed the hyperparameter and term-size of each algorithm, and the top 5 performances are shown in Table 10. Our proposed method, i.e., condensing a variable length of system call traces into a fixed-size of numerical vector using RNN-DAE, yielded near best performance compared with the benchmark methods.

V. CONCLUSION

After the advent of the growth of computer networks and systems, the importance of early detection of cyber attack has emerged. Previous methods for detecting cyber attacks were mainly based on rule-based pattern matching techniques, which are highly dependent on the prior knowledge of experts. To overcome the limitation of pattern matching methods, several machine learning-based approaches have been developed. However, those studies used classification algorithms, which require a sufficient number of labeled attack examples. Since it is almost impossible to obtain a massive amount of labeled training dataset, those studies have an intrinsic limitation to be adopted by real systems. In this study, we propose an unsupervised machine learning-based system anomaly detection framework which does not require a labeled dataset to train the model. RNN-AE and RNN-DAE are used to transform variable lengths of system call traces fixed-sized numeric vectors which preserve sequential information. Based on these feature vectors, anomaly detection model is trained and unsupervised and supervised machine learning-based algorithms were applied to detect abnormal system call traces. Then, anomaly detection algorithms such as IF are used to detect suspicious activities.

Experimental results show the good performance of the proposed log embedding method with unsupervised anomaly detection model yielding at most 0.8828 in terms of area under the receiver operating characteristic curve (AUROC), which can be used in practice. These performances can be improved to at most 0.9850 in terms of AUROC by transferring to supervised classification models once a sufficient number of labeled attack logs are obtained. Despite the affirmative experimental results, there are is a limitation of current study which lead us to future research directions. We only used the system call information to detect abnormal activities. However, the system call traces only reflect some parts of users, one can built a better detection model if various attributes on real systems such as system call, API information, and multiple artifacts considering network, registry, and process, are taken into account.

REFERENCES

[1] M. Abomhara and G. M. Køien, "Cyber security and the Internet of Things: Vulnerabilities, threats, intruders and attacks," *J. Cyber Secur. Mobility*, vol. 4, no. 1, pp. 65–88, May 2015.
[2] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2018, pp. 108–116.
[3] D. D. Kshirsagar, S. S. Sale, D. K. Tagad, and G. Khandagale, "Network intrusion detection based on attack pattern," in *Proc. 3rd Int. Conf. Electron. Comput. Technol.*, vol. 5, Apr. 2011, pp. 283–286.
[4] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2011, pp. 19–24.
[5] A. Kumar, H. C. Maurya, and R. Misra, "A research paper on hybrid intrusion detection system," *Int. J. Eng. Adv. Technol.*, vol. 2, pp. 895–2249, Apr. 2013.
[6] P. Kabiri and A. A. Ghorbani, "Research on intrusion detection and response: A survey," *IJ Netw. Secur.*, vol. 1, no. 2, pp. 84–102, 2005.
[7] R. Koch, M. Golling, and G. D. Rodosek, "Towards comparability of intrusion detection systems: New data sets," in *Proc. TERENA Netw. Conf.*, vol. 7, 2014.
[8] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1188–1196.
[9] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2013, pp. 4487–4492.
[10] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava, and P.-N. Tan, "Data mining for network intrusion detection," in *Proc. NSF Workshop Next Gener. Data Mining*, 2002, pp. 21–30.
[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. for Secur. Defense Appl.*, Jul. 2009, pp. 1–6.
[12] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD," in *Proc. 11th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, Aug. 2014, pp. 978–982.
[13] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD," in *Proc. 11th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*. Xiamen, China: Springer, Aug. 2014, pp. 542–549.
[14] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.
[15] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of SVM and ANN for intrusion detection," *Comput. Oper. Res.*, vol. 32, no. 10, pp. 2617–2634, Oct. 2005.
[16] J. Zhang and M. Zulkernine, "Anomaly based network intrusion detection with unsupervised outlier detection," in *Proc. IEEE Int. Conf. Commun.*, vol. 5, Jun. 2006, pp. 2388–2393.
[17] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, Apr. 2014.
[18] B. Borisaniya and D. Patel, "Evaluation of modified vector space representation using ADFA-LD and ADFA-WD datasets," *J. Inf. Secur.*, vol. 6, no. 3, p. 250, 2015.
[19] W. Haider, J. Hu, and M. Xie, "Towards reliable data feature retrieval and decision engine in host-based anomaly detection systems," in *Proc. IEEE 10th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2015, pp. 513–517.
[20] M. Al-Zewairi, S. Almajali, and A. Awajan, "Experimental evaluation of a multi-layer feed-forward artificial neural network classifier for network intrusion detection system," in *Proc. Int. Conf. New Trends Comput. Sci. (ICTCS)*, Oct. 2017, pp. 167–172.

[21] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1595–1598.
[22] W.-H. Lin, H.-C. Lin, P. Wang, B.-H. Wu, and J.-Y. Tsai, "Using convolutional neural networks to network intrusion detection for cyber threats," in *Proc. IEEE Int. Conf. Appl. Syst. Invention (ICASI)*, Apr. 2018, pp. 1107–1110.
[23] T. Kim, S. C. Suh, H. Kim, J. Kim, and J. Kim, "An encoding technique for CNN-based network anomaly detection," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2960–2965.
[24] W. Anani and J. Samarabandu, "Comparison of recurrent neural network algorithms for intrusion detection based on predicting packet sequences," in *Proc. IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, May 2018, pp. 1–4.
[25] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang, "An intelligent network attack detection method based on RNN," in *Proc. IEEE 3rd Int. Conf. Data Sci. Cyberspace (DSC)*, Jun. 2018, pp. 483–489.
[26] F. Jiang, Y. Fu, B. B. Gupta, Y. Liang, S. Rho, F. Lou, F. Meng, and Z. Tian, "Deep learning based multi-channel intelligent attack detection for data security," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 2, pp. 204–212, Apr. 2020.
[27] X. Zhang and J. Chen, "Deep learning based intelligent intrusion detection," in *Proc. IEEE 9th Int. Conf. Commun. Softw. Netw. (ICCSN)*, May 2017, pp. 1133–1137.
[28] G. Andresini, A. Appice, N. Di Mauro, C. Loglisci, and D. Malerba, "Exploiting the auto-encoder residual error for intrusion detection," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS PW)*, Jun. 2019, pp. 281–290.
[29] G. Andresini, A. Appice, N. D. Mauro, C. Loglisci, and D. Malerba, "Multi-channel deep feature learning for intrusion detection," *IEEE Access*, vol. 8, pp. 53346–53359, 2020.
[30] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, "One class support vector machines for detecting anomalous windows registry accesses," in *Proc. Workshop Data Mining Comput. Secur.*, 2003.
[31] M. A. Kabir and X. Luo, "Unsupervised learning for network flow based anomaly detection in the era of deep learning," in *Proc. IEEE 6th Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, Aug. 2020, pp. 165–168.
[32] T. Scheepers, E. Kanoulas, and E. Gavves, "Improving word embedding compositionality using lexicographic definitions," in *Proc. World Wide Web Conf.*, 2018, pp. 1083–1093.
[33] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. ICML Workshop Unsupervised Transf. Learn.*, 2012, pp. 37–49.
[34] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: http://arxiv.org/abs/1406.1078
[35] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.
[36] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 1096–1103.
[37] G. Lample, A. Conneau, L. Denoyer, and M. Ranzato, "Unsupervised machine translation using monolingual corpora only," 2017, *arXiv:1711.00043*. [Online]. Available: http://arxiv.org/abs/1711.00043
[38] D. Aswath, S. T. Ahmed, J. D'Cunha, and H. Davulcu, "Boosting item keyword search with spreading activation," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Sep. 2005, pp. 704–707.
[39] R. Albatal and S. Little, "Empirical exploration of extreme SVM-RBF parameter values for visual object classification," in *Proc. Int. Conf. Multimedia Modeling*. Cham, Switzerland: Springer, 2014, pp. 299–306.
[40] P. Dhivya and S. Vasuki, "Wavelet based MRI brain image classification using radial basis function in SVM," in *Proc. 2nd Int. Conf. Trends Electron. Informat. (ICOEI)*, May 2018, pp. 1–9.
[41] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.
[42] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Rec.*, vol. 29, no. 2. New York, NY, USA: ACM, 2000, pp. 93–104.

CZANGYEOB KIM received the B.S. degree in computer science from Kookmin University, and the M.S. degree in information security from Sungkyunkwan University. He is currently pursuing the Ph.D. degree with the School of Industrial Management Engineering, Korea University, Republic of Korea. His research interests include developing machine learning algorithms and applying them to solve engineering problems in cybersecurity, computer networks, and healthcare.

MYEONGJUN JANG received the B.S. and M.Eng. degrees in industrial engineering from Korea University. He is currently pursuing the D.Phil. degree with the Department of Computer Science, University of Oxford. He worked as the NLP Scientist and Engineer with the AI Center, SK Telecom, South Korea. His research interests include developing explainable and confidential language models, and low-resource language learning.

SEUNGWAN SEO received the B.S. degree in information and technology management from SeoulTech. He is currently pursuing the Ph.D. degree in industrial and management engineering with Korea University, Seoul, Republic of Korea. His research interests include developing machine learning (especially deep learning) algorithms for unstructured data, such as images and text.

KYEONGCHAN PARK is currently pursuing the Ph.D. degree in industrial and management engineering with Korea University, Seoul, Republic of Korea. His research interests include machine learning and deep learning. Especially developing uncertainty estimation ability of deep learning model.

PILSUNG KANG received the B.S. and Ph.D. degrees in industrial engineering from Seoul National University. He is currently an Associate Professor with the School of Industrial Management Engineering, Korea University, Republic of Korea. His research interests include developing machine learning algorithms for both structured data and unstructured data (image, video, and text) and applying them to solve engineering and business problems, such as fault classification in manufacturing, abnormal behavior detection from system logs, and sentiment classification from news and review texts.