An Empirical Study on Network Anomaly Detection using Convolutional Neural Networks

Donghwoon Kwon* Kathiravan Natarajan*, Sang C. Suh*, Hyunjoo Kim†, Jinoh Kim*

Texas A&M University, Commerce, TX 75428, USA *
ETRI, Gajeong-dong, Yuseong-gu, Daejeon, 34129, Korea †
Email: donghwoon.kwon@tamuc.edu, sunkathirav@gmail.com, sang.suh@tamuc.edu,
hjookim@etri.re.kr, jinoh.kim@tamuc.edu

***Abstract*—Recently, anomaly detection plays a vital role in many areas such as bank fraud, medical examinations, etc. to identify unexpected patterns or events. In particular, this technique in conjunction with deep learning has been applied to network anomaly detection to improve performance. In this research, we establish three different Convolution Neural Network (CNN) architectures based on structural scalability (i.e. shallow CNN, moderate CNN, and deep CNN). Three public traffic datasets are used to evaluate performance of the models, and the shallow CNN model mostly shows higher detection accuracy than other two models. Three models also show a relatively competitive detection accuracy compared to conventional machine learning classifiers. However, experimental results confirm that all three models may have variances in terms of detection accuracy. Furthermore, performance of the shallow CNN model is compared to other deep learning models based on Fully Connected Neural Networks (FCN), Variatinal AutoEncoder (VAE), and Sequence-to-Sequence structure with LSTM (Seq2Seq-LSTM) evaluated in our previous research. We observed that the shallow CNN model occasionally outperforms VAE models, but does not work better than FCN and Seq2Seq-LSTM.**

***Index Terms*—Network Anomaly Detection, Deep Learning, CNN**

I. Introduction

The concept of deep learning has been widely employed in many areas such as speech recognition, language modeling, network security, etc. due to its advantage of dealing with high dimensional data with non-linear and unbalanced properties, as well as significant improvement in accuracy [1] [2]. Conventional machine learning models, e.g. Naive Bayes, Support Vector Machine (SVM), Random Forest, etc., for network anomaly detection have been proposed in many studies [2] [3] [4] [5], but one of the main limitations is that only well-balanced traffic datasets are adopted to evaluate model performance. Hence, this limitation injects motivation into development of the deep learning model for non-linear and unbalanced datasets.

Our prior research successfully proposed a set of deep learning models, e.g. Fully Connected Network (FCN), Variational Auto Encoder (VAE), and Sequence to Sequence model with Long Short-Term Memory (Seq2Seq-LSTM), for network anomaly detection using two public traffic datasets such as NSL-KDD (balanced) and Kyoto Honeypot (unbalanced) [6]. Among those proposed models, the Seq2Seq-LSTM model showed the best detection accuracy, which was 99% of binary classification accuracy on both datasets. Yet, previous research arose our curiosity with respect to performance of the Convolutional Neural Network (CNN) model because this methodology was not employed for network anomaly detection as one of the deep learning models. For this reason, this work is aimed at evaluating performance of a set of CNN models for network anomaly detection. In this paper, we present our initial evaluation results conducted with the data sets of NSL-KDD, Kyoto Honeypot, and MAWILab traffic datasets.

This paper is organized as follows: Section 2 provides a brief description of the datasets and literature review; Section 3 presents a methodology of the CNN model for network anomaly detection; Evaluation results with two traffic datasets are discussed in Section 4; and Section 5 describes conclusions and future work.

II. Related Work

In this section, we briefly discuss network traffic datasets and aspects of network anomaly detection using the CNN model from the literature.

A. Description of the NSL-KDD Dataset

The NSL-KDD dataset is a refined version of the KDDCup 1999 dataset by reducing duplicated records [2] [7]. There are four files in this dataset, i.e. Train+, Train-, Test+, and Test-, and each file is composed of forty two features including label information. Note that forty two features are composed of categorical and numerical values for each record. Labels are categorized as either normal or four attack groups, e.g. *DOS*, *U2R*, *R2L*, and *PROBE*, and those attack groups can be also grouped into anomalies. Hence, there are two possible classification experiments with the dataset: binary classification or multi classification.

B. Description of the Kyoto Koneypot Dataset

Kyoto Honeypot is a daily-based evaluation dataset for network anomaly detection obtained from diverse honeypots from November 2006 by capturing real network traffic [10]. There are a total of twenty four features: fourteen conventional features and ten additional features. In addition, labels in

this dataset indicates as follows: 1 for normal sessions, -1 for known attacks, and -2 for unknown attacks. Yet, since unknown attacks are extremely low causing an issue not to be correctly detected by machine learning models, binary classification is more effective; that is, 0 for normal sessions and 1 for known and unknown attacks.

C. Description of the MAWILab Dataset

MAWI is publicly available and a collection of network traffic traces captured everyday at 15 minutes intervals on a link between Japan and USA since 2001 [8] [9]. The MAWILab dataset in this archive has one important characteristic for the procedure of performance testing; in other words, every traffic flow in the MAWILab framework is classified by the probability-based labels obtained by combining the outputs of four network anomaly detectors, i.e. Kullback-Leibler (KL), Gamma, Hough, and Principal Component Analysis (PCA), to indicate the presence of an anomaly. Those labels are used as a reference for testing model performance. Note that there are four traffic categories: anomalous, suspicious, notice, and benign, and anomalies in each trace are identified by traffic features like IP address, port information, etc.

D. Network Anomaly Detection with CNN

The work in [11] utilized a simple CNN model with multiple layers and hybrid CNN model with Recurrent Neural Networks (RNNs), Long Short-Term Memory Units (LSTMs), and Gated Recurrent Units (GRUs) for network anomaly detection with the KDDCup 99 dataset. There were three layers consisting of an input layer, a hidden layer, and an output layer, and the hidden layer had one or more CNN layer(s) followed by traditional Feed Forward Networks (FFNs) or three other deep learning models as mentioned above. The proposed model was implemented in TensorFlow, and hyper parameters used for model training and experiments were as follows:

- inputs = 41 * 1
- filters = 16, 32, and 64
- filter length = 3 and 5
- learning rate = the range of 0.01 through 0.5
- epochs = up to 1,000

In the experiments for binary classification with the KDDCup 99 dataset, both 1-layer and 2-layer CNN models showed 99% of the highest F-measure accuracy. Yet, the 3-layer CNN model with LSTMs showed 98.7% of the highest accuracy in the experiments for multi classification. Note that using KDDCup99 dataset can expect much higher detection accuracy due to redundant records [12].

In [13], authors proposed an image conversion method of the NSL-KDD dataset using the CNN model for intrusion detection. The first step in this model was feature extraction. A total of forty one extracted features were passed to the second step for data preprocessing. Three categorical features, *protocol_type*, *flags*, and *service*, were converted into binary vectors through one-hot encoding, and numerical features were normalized by standard scaler. By doing so, the dataset was converted into binary vectors with 464 dimensions, and then these vectors were converted into 8*8 gray-scale pixel images. In the last step, two popular CNN models, ResNet 50 and GoogLeNet, were employed to identify the category of image conversion. In addition, the research framework was implemented in TensorFlow.

The number of epochs and batch size, as well as the information of the optimizer and loss function to train both models was as follows:

- epochs: 100 for both models
- batch size: 256 for ResNet 50 and 62 for GoogLeNet
- optimizer: gradient descent
- loss function: cross entropy

The NSL-KDD Train+ was used for model training, and both NSL-KDD Test+ and Test- were adopted for experiments. According to experimental results, ResNet 50 and GoogLeNet with the NSL-KDD Test- showed 89.85% and 90.01% of the highest F1 score, respectively.

III. Proposed Method

According to literature review and limitations in our previous research, we decided to employ the CNN model. Basically, the CNN model is very powerful when it is used for image datasets. Its fundamental idea is that this model is composed of one or more convolution and pooling layer(s) followed by one or more FCN layer(s) [14] [15] [16]. The general architecture of the CNN model in this research has a 1D convolution layer, 1D pooling layer, and FCN layer. Note that we call this CNN architecture the *shallow* CNN model because there is only one convolution, pooling, and FCN layer. Since we have a plan to build deeper CNN models, we call them *moderate* and *deep* CNN models, respectively. Detailed information with respect to each CNN model is described in Table I on the next page.

The reason for using a 1-dimension (1D) convolution layer is that while the CNN model takes image datasets as a form of 2D inputs, it takes both traffic datasets as a form of 1D inputs. In addition, each dataset to be fed into the Convolution 1D (Conv1D) layer is pre-processed as follows:

- NSL-KDD dataset: numerical features are normalized by the Min-Max scaler, and categorical features are converted into a set of dummy numerical values through one-hot encoding. By doing so, a total of 42 features become 124 features including labels. Note that there are 122 features excluding labels.
- Kyoto Honeypot dataset: 5 features out of 24 features are dropped: *start_time*, *source_port_number*, *source_ip_address*, *destination_port_number*, and *destination_ip_address*. One-hot encoding is applied to categorical features so that a total of 19 features become 47 features including labels. There are 45 features excluding labels. The extracted features are then normalized by the Min-Max scaler.
- MAWILab dataset: 5 features out of 29 features are extracted: *pro*, *packets*, *bytes*, *durat*, and *status*. One-hot encoding is applied to the categorical feature which is the status feature so that a total of 5 features become 6

TABLE I
ARCHITECTURE INFORMATION OF EACH CNN MODEL

Models	Conv1D Layer	Max Pooling1D Layer	Flatten	FCN Layer	Softmax Layer	Others
Shallow CNN	1 Conv1D Layer (64 filters, filter size: 3*1)	1 max pooling layer in the Conv1D layer, Stride: 2	outputs: 3904 for NSL-KDD, 1408 for Kyoto, and 192 for MAWILab	1 FCN Layer (Neurons: 64)	outputs: 2	batch normalization, dropout=0.5, loss=binary_crossentropy, optimizer=Adam, epochs=10 and 20 for NSL-KDD Train+ and Train-, respectively, 20 for MAWILab, and 10 for Kyoto Honeypot
Moderate CNN	2 Conv1D Layers (64 and 128 filters, filter size: 3*1)	1 max pooling layer in the second Conv1D layer, Stride: 2	outputs: 7808 for NSL-KDD, 2816 for Kyoto, and 384 for MAWILab	2 FCN Layers (Neurons: 64 and 32)	outputs: 2	same as above
Deep CNN	3 Conv1D Layers (64, 128, and 256 filters, filter size: 3*1)	2 max pooling layers in the second and third Conv1D layer, Stride: 2	outputs: 7680 for NSL-KDD, 2816 for Kyoto, and 512 for MAWILab	3 FCN Layers (Neurons: 64, 32, and 16)	outputs: 2	same as above

features. These expanded data are then normalized by the Min-Max scaler.

Once a vector of each pre-processed input data are given to the Conv1D layer, feature maps are generated by filters in conjunction with a stride and padding followed by activation functions such as Rectifier Linear Unit (ReLu), tanh, etc. [15]. The feature map can be calculated by the ReLu or tanh activation as follows [11] [15]:

- when the ReLu non-linear activation is employed:

$$h_i^k = max(w^k x_i, 0) \tag{1}$$

- when the tanh non-linear activation is employed:

$$h_i^k = tanh(w^k x_i) \tag{2}$$

where h^k denotes the *k*th feature map at a given layer, *i* is the index in the feature map, x_i indicates the input, and w^k denotes the weights. Among these two non-linear activations, the ReLu activation is chosen for calculating the feature map in this research. Furthermore, the number and size (length) of a filter is decided by our subjective intuition, and a padding is set to *same* to make outputs of the convolutional layer same as inputs.

There are two different types of pooling layers: average pooling layer and max pooling layer, denoted as [15] [17]:

- average pooling

$$f_{avg}(x) = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3}$$

- max pooling

$$f_{max}(x) = max(x_i) \tag{4}$$

where *x* denotes a vector of input data with activation values and *N* indicates a local pooling region. Among them, the max pooling layer is adopted because it is used as the default pooling layer more frequently than the average pooling layer in the CNN model [17]. It is possible to reduce the size of the feature maps through the max pooling layer in conjunction with a stride by picking up the feature which has the highest value [11], and we set a stride to 2 as a default value for fast computation.

Lastly, once the flattened output of the max pooling layer is fed to the FCN, the model performs training with the ReLu activation, batch normalization, and dropout. Both batch normalization and dropout are applied to each Conv1D and FCN layer to improve performance. We set 0.5 to the dropout rate, and the number of neurons in a FCN layer for each model is also determined by our subjective intuition. Hyper parameters like the number of neurons and epochs are shown in Table I.

IV. EVALUATION

In this section, we report the experimental results performed with the three datasets: NSL-KDD ("balanced"), Kyoto Honeypot ("unbalanced"), and MAWILab ("balanced"). Prior to that, a brief description of the experimental setup is explained in the next section.

A. Experimental Setup

To build CNN models and evaluate them, we considered Intel core i7-3770 3.4GHz with 8GB RAM workstation. Each CNN model was implemented in Python Keras package, and both traffic datasets were pre-processed in Tensorflow. All pairs of NSL-KDD data and a subset of MAWILab and Kyoto Honeypot were used for experiments. We chose a single day MAWILab data consisting of multiple flows and used one flow data for training and the rest of flow data for testing. We intentionally had a two-year gap Honeypot data for training and testing. Note that the Kyoto Honeypot dataset is highly skewed (97% of the records have attack labels), and we used one training data and three testing data.

B. Experimental results with NSL-KDD

To evaluate performance of each CNN model with the NSL-KDD dataset, we employed *F-measure*, also known as *F1-*

Score. *Training time* for training complexity was not considered this time because deeper CNN models take longer training time than shallow one from a common-sense point of view. Figure 1 depicts the measured experimental results for the NSL-KDD dataset.

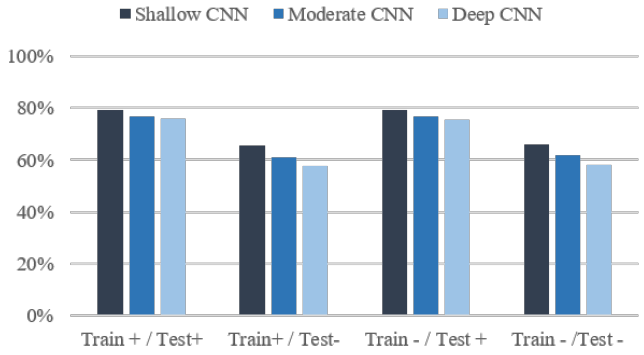

Fig. 1. Experimental Results for the NSL-KDD Dataset

Among three CNN models, the shallow CNN model outperformed two other CNN models even if this model took less training time than both models. Based on these experimental results, we compared the shallow CNN model to four other deep learning models, i.e. FCN, VAE-Label, VAE-Softmax, and Seq2Seq-LSTM, which we built in previous research [6].

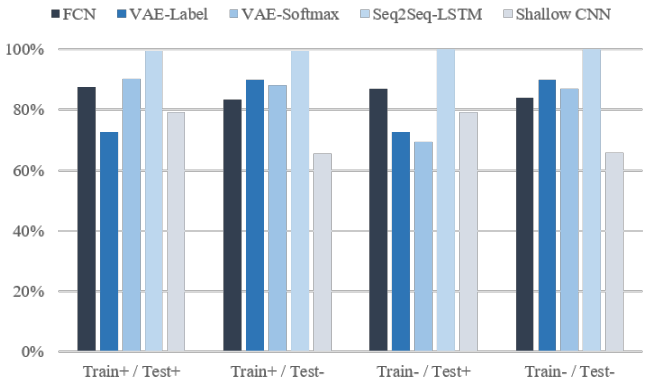

Fig. 2. Comparison Analysis of the Deep Learning Models

As shown in Figure 2 above, performance of the shallow CNN model with a combination of Train+ / Test+ was higher than the VAE-Label, but its performance was lower than other three models with combinations of Test+ / Test+ and Train- / Test-. In addition, the shallow CNN model showed higher performance than both VAE models with a combination of Train- / Test+. However, according to the results shown in Table II, our proposed method had a relatively competitive performance compared to conventional machine learning classifiers [13] [18] in terms of testing accuracy, not *F-measure* because the KDDCup99 dataset was used for evolution in those studies, whereas our models used the NSL-KDD dataset.

TABLE II
EXPERIMENTAL RESULTS OF MODEL TRAINING AND TESTING

Models	Accuracy on Test+	Accuracy on Test-
J48	81.05%	63.97%
Naive Bayes	76.56%	55.77%
NB Tree	82.02%	66.16%
Random Forest	80.67%	62.26%
MLP	77.41%	57.34%
SVM	69.52%	42.29%
Shallow CNN	79.44%	61.59%
Moderate CNN	78.33%	60.30%
Deep CNN	76.79%	55.58%

C. Experimental results with Kyoto-Honeypot

As mentioned earlier, the Kyoto-Honeypot dataset is very skewed. For this reason, *F-measure* is not reliable due to the fact that it could lead to a critical bias caused by the high degree of skewness. Hence, a measure of Matthew Correlation Coefficient (MCC) was employed to evaluate the quality of binary classification. Note that the interpretation of the MCC value is as follows: 1.0 for good, 0.0 for random, and -1.0 for poor. As shown in Figure 3, the Deep CNN model

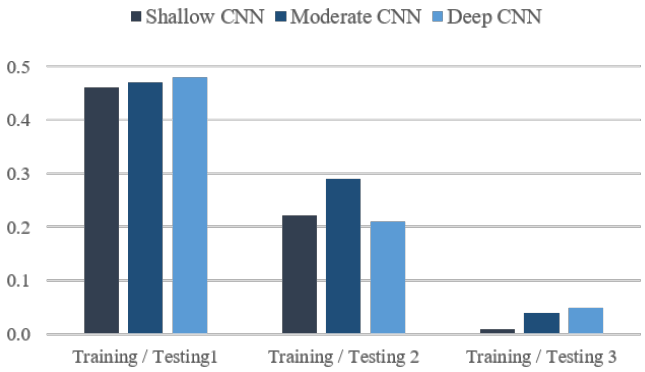

Fig. 3. Experimental Results for the Kyoto Honeypot in Terms of MCC

showed higher MCC scores than two other CNN models except a combination of the second evaluation data. Hence, we conducted experiments again to see why the deep CNN model did not outperform two models consistently. The second experiments showed that the deep CNN model outperformed two other CNN models same as the first experiments, but it still showed inconsistent evaluations with a combination of the second evaluation data. For reference, the shallow CNN model showed the highest MCC score with a combination of the second evaluation data. Therefore, we could conclude that there were variances depending upon a dataset.

D. Experimental results with MAWILab

Since performance of the CNN models showed variances with the unbalanced dataset, this point aroused another question: "Does the shallow CNN model show higher performance than other two CNN models consistently with balanced datasets?" To find an answer for this question, we conducted experiments with the MAWILab dataset. The proportion of *Normal* and *Anomaly* in each flow of this dataset is shown in Table III.

TABLE III
MAWILAB DATASET ON AUGUST 27, 2017

Flow	# data points	# normal	# anomaly	% anomaly
Flow 001	407,807	291,488	116,319	28.5%
Flow 002	472,654	327,413	145,241	30.7%
Flow 003	423,984	261,426	162,558	38.3%
Flow 004	425,994	300,759	125,235	29.4%

As mentioned earlier, we used the flow001 data for training and the rest of flow data for testing. In addition, *F-measure* was employed again to evaluate performance like the NSL-KDD dataset, and experimental results are shown in Table IV. Experimental results were interesting. We expected that

TABLE IV
EXPERIMENTAL RESULTS WITH THE MAWILAB DATASET

Model	Flow 001 / Flow 002	Flow 001 / Flow 003	Flow 001 / Flow 004
Shallow CNN	65.44%	59.27%	61.33%
Moderate CNN	65.41%	67.66%	59.76%
Deep CNN	65.45%	67.86%	56.76%

the shallow CNN model would show higher performance than other two models consistently, but its performance was lower than other ones with a combination of the Flow 001 and Flow 003 data. Hence, we found an important fact that CNN models could also have variances with balanced datasets in terms of performance.

V. CONCLUSION

In this work, we established a set of CNN models and evaluated performance with three public traffic datasets. The shallow CNN model mostly outperformed both moderate and deep CNN models, and all three CNN models yielded very competitive detection accuracy compared to conventional machine learning classifiers. Yet, those models did not outperform other deep learning models such as the FCN and Seq2Seq-LSTM. Furthermore, the established CNN models showed variances in terms of detection accuracy with both balanced and unbalanced datasets. In this initial work, we evaluated basic CNN models. We next plan to extend the models with some optimizations to see if there would be performance improvement.

ACKNOWLEDGMENT

We thank Ritesh K. Malaiya for providing pre-processed datasets and sharing the experimental results of the deep learning models.

REFERENCES

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

[2] S. Baek, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim, "Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks," in *Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on*. IEEE, 2017, pp. 205–210.

[3] P. Nevlud, M. Bures, L. Kapicak, and J. Zdrálek, "Anomaly-based network intrusion detection methods," *Advances in Electrical and Electronic Engineering*, vol. 11, no. 6, p. 468, 2013.

[4] W. Hu, Y. Liao, and V. R. Vemuri, "Robust anomaly detection using support vector machines," in *Proceedings of the international conference on machine learning*, 2003, pp. 282–289.

[5] R. Primartha and B. A. Tama, "Anomaly detection using random forest: A performance revisited," in *Data and Software Engineering (ICoDSE), 2017 International Conference on*. IEEE, 2017, pp. 1–6.

[6] R. Malayiya, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim, "An empirical evaluation of deep learning for network anomaly detection," in *International Conference on Computing, Networking and Communications*. IEEE, 2018.

[7] L. Dhanabal and S. Shantharajah, "A study on nsl-kdd dataset for intrusion detection system based on classification algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 446–452, 2015.

[8] J. Mazel, R. Fontugne, and K. Fukuda, "Visual comparison of network anomaly detectors with chord diagrams," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 473–480.

[9] C. Callegari, S. Giordano, and M. Pagano, "Statistical network anomaly detection: An experimental study," in *International Conference on Future Network Systems and Security*. Springer, 2016, pp. 12–25.

[10] J. Song, H. Takakura, and Y. Okabe, "Description of kyoto university benchmark data," *Available at link: http://âĂŃ www. âĂŃ takakura. âĂŃ com/âĂŃ Kyoto_âĂŃ data/âĂŃ BenchmarkData-Description-v5. âĂŃ pdf.[Accessed on 15 March 2016]*, 2006.

[11] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on*. IEEE, 2017, pp. 1222–1228.

[12] M. M. U. Chowdhury, F. Hammond, G. Konowicz, C. Xin, H. Wu, and J. Li, "A few-shot deep learning approach for improved intrusion detection," in *Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017 IEEE 8th Annual*. IEEE, 2017, pp. 456–462.

[13] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *International Conference on Neural Information Processing*. Springer, 2017, pp. 858–866.

[14] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4277–4280.

[15] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[16] S. Hijazi, R. Kumar, and C. Rowen, "Using convolutional neural networks for image recognition," *Cadence Design Systems Inc.: San Jose, CA, USA*, 2015.

[17] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *Artificial Intelligence and Statistics*, 2016, pp. 464–472.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 2009, pp. 1–6.