Insider Threat Detection with Deep Neural Network

Fangfang Yuan[1,2,3], Yanan Cao[1,3], Yanmin Shang[1,3], Yanbing Liu[1,3(✉)], Jianlong Tan[1,3] and Binxing Fang[4]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] National Engineering Laboratory for Information Security Technologies, Beijing, China
[4] Institute of Electronic and Information Engineering of UESTC in Guangdong, Dongguan, Guangdong

{yuanfangfang, caoyanan, shangyanmin, liuyanbing, tanjianlong}@iie.ac.cn, fangbx@cae.cn

Abstract. Insider threat detection has attracted a considerable attention from the researchers and industries. Existing work mainly focused on applying machine-learning techniques to detecting insider threat. However, this work requires "feature engineering" which is difficult and time-consuming. As we know, the deep learning technique can automatically learn powerful features. In this paper, we present a novel insider threat detection method with Deep Neural Network (DNN) based on user behavior. Specifically, we use the LSTM-CNN framework to find user's anomalous behavior. First, similar to natural language modeling, we use the Long Short Term Memory (LSTM) to learn the language of user behavior through user actions and extract abstracted temporal features. Second, the extracted features are converted to the fixed-size feature matrices and the Convolutional Neural Network (CNN) use these fixed-size feature matrices to detect insider threat. We conduct experiments on a public dataset of insider threats. Experimental results show that our method can successfully detect insider threat and we obtained AUC = 0.9449 in best case.

Keywords: Insider Threat, Anomaly Detection, Deep Learning, Network Security.

1 Introduction

Insider threat is becoming a serious security challenge for many organizations. It is generally defined as malicious actions performed by an insider in a secure environment, often causing system sabotage, electronic fraud and information theft. Hence, it is potentially harmful to individuals, organizations and state security. Recently, insider threat detection has attracted considerable attention in both academic and industrial community.

Insider threat detection becomes an extremely complex and challenging task. The reasons are as follows. First, insiders do unauthorized things by the use of their trusted access. Hence, external network security devices (intrusion detection, firewalls, and anti-virus) cannot detect them. Second, insider attack manifests in various forms, such as a disgruntled employee planting a logic bomb to disrupt systems, stealing intellectual property for personal gain, etc. The diversity of insider attack increases the complexity of insider threat detection. The last but not the least, insider threat often performed by insiders during working hours, causing insider's anomalous behaviors scattered in large amounts of normal working behaviors. Therefore, it increases the difficulty of insider threat detection.

The key of insider threat detection is to model a user's normal behavior to detect anomalous behavior. Much work has been proposed to address the issue [1-2]. They aggregate all the actions of a user in one day to represent the user's behavior in the same day. However, the anomalous behavior happening within one day may be missed. For example, a user logs on to his assigned computer after hours and uploads data to wikileaks.org. We argue that using user action sequences for each user is very important in detecting insider threat.

To address this problem, we propose a novel insider threat detection method to detect whether user behavior is normal or anomalous. Specifically, it is not efficient that we directly use the LSTM to classify the user action sequence, because the output of the LSTM only contains a single bit of information for every sequence. Instead, we use the trained LSTM to predict next user action, and use a series of hidden states of the LSTM model to generate a fixed-size feature matrix that is given to the CNN classifier. The LSTM can better capture the long term temporal dependencies on user action sequence, because hidden units of the LSTM potentially record temporal behavior patterns.

To summarize, in this paper, we make the following contributions:

(1) We present a novel insider threat detection method with LSTM and CNN based on user behavior.

(2) We use the LSTM to learn the language of user behavior through user actions and extract abstracted temporal features which are the input of the CNN classifier.

(3) Experimental results on a public dataset of insider threats show that our proposal can successfully detect insider threat and we obtained AUC = 0.9449 in best case.

The rest of this paper is organized as follows. We summarize the related work in Section 2, and give a detailed description of our insider threat detection method in Section 3. Implementation details and experimental results for this work are shown in Section 4. Finally, we conclude the paper's work in Section 5.

2 Related Work

Related work falls into two main categories, insider threat detection and deep neural network.

Insider Threat Detection: The problem of insider threat detection is usually framed as an anomaly detection task. A comprehensive and structured overview of anomaly detection techniques was provided by Chandola et al. [3]. They defined that the purpose of anomaly detection is finding patterns in data which did not conform to the expected behavior. The key problem of anomaly detection is how to model a user's normal behavior profile. A lot of research work has been proposed to develop anomaly detection, especially machine learning.

Early work on anomaly detection based on user command proposed by Davison et al. [4] and Lane et al. [5]. They examine user command sequences and compute the match degree of a current command pattern with the historical command pattern to classify user behavior as normal or anomalous.

After that, anomaly detection begins to take advantage of machine learning techniques, such as Naive Bayes [6], Eigen Co-occurrence Matrix (ECM) [7], One-Class Support Vector Machine (OC-SVM) [8] and Hidden Markov [9]. Schonlau et al. compared the performance of six masquerade-detection algorithms on the data set of "truncated" UNIX shell commands for 70 users and experimental results revealed that no single method completely dominated any other. Maxion et al. [6] applied the Naive Bayes classifier to the same data set [17], inspired by Naive Bayes text classification. They also provided a thorough and detailed investigation of classification errors of the classifier in [18]. Oka et al. [7] argued that the causal relationship embedded in sequences of events should be considered when modeling a user's profile. They developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM) and extracted the causal relationships embedded in sequences of commands to supplement user behavior model. Salem et al. [19] evaluated the accuracy performance of the nine methods mentioned above using the Schonlau dataset, but the results revealed that their detection rates were not high. Szymanski et al. [8] used an OC-SVM classifier for insider threat detection. However, the approach needed mixing user data and it was hard to implement in a real-world setting. Rashid et al. [9] proposed an approach to insider threat detection by the use of Hidden Markov. They used Hidden Markov to model user's normal behavior via user actions and regarded deviations from the normal behavior as anomalous behavior. The effectiveness of the method is highly impacted by the number of the states. However, the computational cost of the Hidden Markov model increases as the number of states increases.

The works mentioned above make use of machine learning techniques to build a classifier. On one hand, machine learning requires "feature engineering" which is time-consuming and difficult. On the other hand, the classifier is too simple, resulting in a low detection rate.

Deep Neural Network: Recently, deep neural network that can automatically learn powerful features has led to new ideas for anomaly detection. Tang et al. [10] applied the deep learning methodology to build up an anomaly detection system, but the experimental results in the testing phase were not good enough. Veeramachananeni et al. [11] used a neural network auto-encoder to detect insider threat. They aggregated a number of numeric features over a time window and fed these features to an ensemble of anomaly detection methods: Principal Component Analysis, neural networks, and a probabilistic model. However, individual user activity was not explicitly modeled over time. Tuor et al. [2] proposed a deep learning approach to detect anomalous network activity from system logs. They trained Recurrent Neural Networks (RNNs) to recognize characteristic of each user on a network and concurrently assessed whether user behavior is normal or anomalous. While this method aggregates features over one day for individual users, it is possible to miss anomalous behavior happening within one day. Instead, our model is trained using user action sequences with DNN. The actions that a user takes over a period of time on a system can be modeled as a sequence. The action sequences of user's normal behavior are seen often or on a usual basis. Observed action sequences deviated from those normal action sequences are regarded as anomalous behavior. Therefore, our model can detect anomalous behavior through user actions and even can detect anomalous behavior happening within one day.

3 Proposed method

In this section, we introduce the details of our insider threat detection method. The proposal applies DNN in two stages. The first stage extracts the abstracted temporal features of user behavior by the LSTM and outputs feature vectors. Then the feature vectors are transformed into fixed-size feature matrices. In the second stage, these fixed-size feature matrices are fed to the CNN to classify them as normal or anomaly.

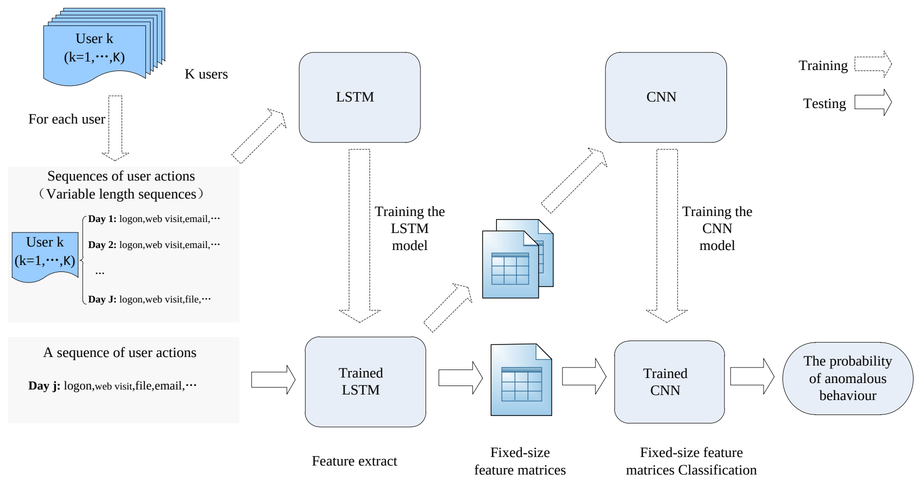

User k (k=1,⋯,K)
K users
For each user
Sequences of user actions (Variable length sequences)
User k (k=1,⋯,K)
Day 1: logon,web visit,email,⋯
Day 2: logon,web visit,email,⋯
…
Day J: logon,web visit,file,⋯
LSTM
Training the LSTM model
CNN
Training the CNN model
Training
Testing
A sequence of user actions
Day j: logon,web visit,file,email,⋯
Trained LSTM
Trained CNN
The probability of anomalous behaviour
Feature extract
Fixed-size feature matrices
Fixed-size feature matrices Classification

Fig. 1. Overview of proposed method

3.1 Overview

The overview of our insider threat detection method is shown in Fig. 1. The individual action (e.g., logging onto an assigned computer afterhours) represents the operation of a user; actions taken by a user in one day represent user behavior. Similar to natural language modeling, an action is corresponding to a word and an action sequence is corresponding to a sentence. For that reason, we attempt to learn the language of user behavior as a new method for detecting insider threat. The LSTM is used to extract the features of user behavior. The CNN uses these features to find anomalous behavior.

Let U $= \{u_1, u_2, \cdots, u_K\}$ be the set of K users. For a user $u_k (1 \le k \le K)$, we can obtain his action sequences over J days, $\mathbf{S} = \left[\mathbf{s}_{u_{k,1}}, \mathbf{s}_{u_{k,2}}, \cdots, \mathbf{s}_{u_{k,J}}\right]$, where $\mathbf{s}_{u_{k,j}} (1 \le j \le J)$ is a vector which denotes the action sequence on the day indexed by j. In the training phase, we first obtain an action sequence $\mathbf{s}_{u_{k,j}}$ that user u_k has performed within the day indexed by j. Second, the action sequence $\mathbf{s}_{u_{k,j}}$ is then fed into the LSTM and the LSTM is trained to construct a feature extractor to obtain the abstracted feature vectors in the deep layer. Third, the feature vectors are transformed into a fixed-size matrix $\mathbf{M}^{u_{k,j}}$. The fixed-size feature matrix potentially contains various abstracted temporal features that represent user behavior. Finally, we use these fixed-size matrices annotated with normal or anomalous to train the CNN. In the testing phase, we evaluate the approach with the trained LSTM and the trained CNN. The detail of each step is described in the following subsections.

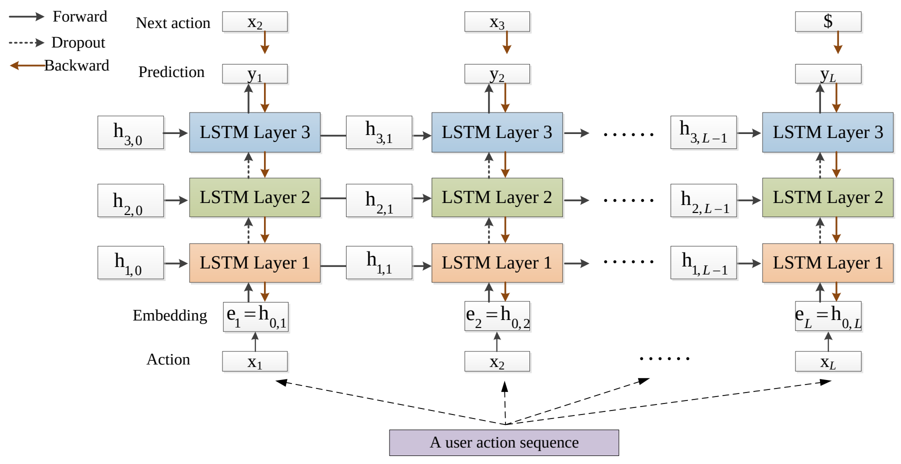


Fig. 2. Flow of LSTM training

3.2 Training LSTM for Feature Extraction

Based on the user action sequences, we construct a feature extractor which can automatically extract abstracted temporal features from each input action sequence. The LSTM consists of an input layer, an embedding layer, three LSTM layers, and an output layer. The flow of the LSTM is shown in Fig. 2.

For user u_k on the day indexed by j, let T be the length of the action sequence, $\mathbf{s}_{u_{k,j}} = [\mathbf{x}_1^{u_{k,j}}, \mathbf{x}_2^{u_{k,j}}, \cdots, \mathbf{x}_T^{u_{k,j}}]$. $\mathbf{x}_t^{u_{k,j}} (1 \le t \le T)$ represents an individual action at time instance t. $\mathbf{h}_{l,t}^{u_{k,j}} (0 \le l \le 3, 1 \le t \le T)$ denotes the hidden state of hidden layer l at time instance t. $\mathbf{y}_t^{u_{k,j}} (1 \le t \le T)$ denotes the output at time instance t. Here we use one-hot encoding to embed the input $\mathbf{x}_t^{u_{k,j}}$ as a vector $\mathbf{e}_t^{u_{k,j}} (1 \le t \le T)$. The one-hot encoding is performed as follows:

1. Creating a dictionary in which IDs and actions are associated with each other, such as logging on an assigned PC after hours is denoted as 1, logging off an assigned PC after hours is denoted as 2, etc.
2. Converting actions to one-hot vectors, which is 1 at the action ID position, and 0 elsewhere.

The LSTM with three hidden layers ($l = 1,2,3$) is described by the following equations:

$$\mathbf{i}_{l,t}^{u_{k,j}} = \sigma\left(\mathbf{W}_l^{(i,x)}\mathbf{h}_{l-1,t}^{u_{k,j}} + \mathbf{W}_l^{(i,h)}\mathbf{h}_{l,t-1}^{u_{k,j}} + \mathbf{b}_l^i\right) \quad (1)$$

$$\mathbf{f}_{l,t}^{u_{k,j}} = \sigma\left(\mathbf{W}_l^{(f,x)}\mathbf{h}_{l-1,t}^{u_{k,j}} + \mathbf{W}_l^{(f,h)}\mathbf{h}_{l,t-1}^{u_{k,j}} + \mathbf{b}_l^f\right) \quad (2)$$

$$\mathbf{o}_{l,t}^{u_{k,j}} = \sigma\left(\mathbf{W}_l^{(o,x)}\mathbf{h}_{l-1,t}^{u_{k,j}} + \mathbf{W}_l^{(o,h)}\mathbf{h}_{l,t-1}^{u_{k,j}} + \mathbf{b}_l^o\right) \quad (3)$$

$$\mathbf{g}_{l,t}^{u_{k,j}} = \tanh\left(\mathbf{W}_l^{(g,x)}\mathbf{h}_{l-1,t}^{u_{k,j}} + \mathbf{W}_l^{(g,h)}\mathbf{h}_{l,t-1}^{u_{k,j}} + \mathbf{b}_l^g\right) \quad (4)$$

$$\mathbf{c}_{l,t}^{u_{k,j}} = \mathbf{f}_{l,t}^{u_{k,j}} \odot \mathbf{c}_{l,t-1}^{u_{k,j}} + \mathbf{i}_{l,t}^{u_{k,j}} \odot \mathbf{g}_{l,t}^{u_{k,j}} \quad (5)$$

$$\mathbf{h}_{l,t}^{u_{k,j}} = \mathbf{o}_{l,t}^{u_{k,j}} \odot \tanh\left(\boldsymbol{c}_{l,t}^{u_{k,j}}\right) \quad (6)$$

Where $\mathbf{h}_{0,t}^{u_{k,j}} = \mathbf{e}_t^{u_{k,j}}$, and $\mathbf{c}_{l,0}^{u_{k,j}}$, $\mathbf{h}_{l,0}^{u_{k,j}}$ are set to zero vector for all $1 \le l \le 3$. $\sigma(\cdot)$ is the sigmoid function and $\odot$ denotes element-wise multiplication. Vector $\mathbf{g}_{l,t}^{u_{k,j}}$ is a hidden representation, vector $\mathbf{i}_{l,t}^{u_{k,j}}$ decides which values to update, vector $\mathbf{f}_{l,t}^{u_{k,j}}$ decides which things to forget, vector $\mathbf{o}_{l,t}^{u_{k,j}}$ decides what to be outputted. 24 weight matrices (**W**) and 12 bias vectors (**b**) are learned parameters.

The LSTM is repeatedly trained using user action sequences. First, we take an input series of user u_k as a vector $\mathbf{A}^{u_{k,j}}=\left[\mathbf{x}_1^{u_{k,j}}, \mathbf{x}_2^{u_{k,j}}, \cdots, \mathbf{x}_T^{u_{k,j}}\right]$. Second, the embedding layer converts the series of actions $\mathbf{A}^{u_{k,j}}$ to one-hot vectors $\mathbf{E}^{u_{k,j}}=\left[\mathbf{e}_1^{u_{k,j}}, \mathbf{e}_2^{u_{k,j}}, \cdots, \mathbf{e}_T^{u_{k,j}}\right]$. Third, we sequentially input each one-hot vector $\mathbf{e}_t^{u_{k,j}}$ to the LSTM and the LSTM outputs prediction $\mathbf{y}_t^{u_{k,j}}$. Finally, we calculate the cross-entropy loss function by comparing prediction $\mathbf{y}_t^{u_{k,j}}$ with answer $\mathbf{x}_{t+1}^{u_{k,j}}$.

In training phase, we apply Dropout [12] to the LSTM in a way that can reduce overfitting. The dropout operator is only applied to the non-recurrent connections. One epoch means that all training user action sequences are inputted to the LSTM. The order of user action sequences is randomized in every epoch. The LSTM training is executed for multiple epochs. After training, we obtain the trained feature extractor. Then we extract the hidden state of the last hidden layer (the third layer in Fig. 2) for every input and obtain a series of feature vectors $\mathbf{H}^{u_{k,j}}=\left[\mathbf{h}_{3,1}^{u_{k,j}}, \mathbf{h}_{3,2}^{u_{k,j}}, \cdots, \mathbf{h}_{3,T}^{u_{k,j}}\right]$.

3.3 Fixed-size Feature Representations

As the designed classifier accepts fixed-size representations and the number of actions differs between user action sequences, we need to construct a fixed-size feature matrix for the series of feature vectors which is provided as input of the CNN.

To deal with this, we decided on a maximal length N^{u_k} and a minimal length n^{u_k} for any action sequence for user u_k. We ignore all sequences whose length are shorter than n^{u_k}. For all sequences with more than N^{u_k} steps, we keep only the first N^{u_k} actions. For all sequences whose length T is between n^{u_k} and N^{u_k}, we pad them with zeros until their lengths reach N^{u_k}. By this way, we can convert the series of feature vectors $\mathbf{H}^{u_{k,j}}=\left[\mathbf{h}_{3,1}^{u_{k,j}}, \mathbf{h}_{3,2}^{u_{k,j}}, \cdots, \mathbf{h}_{3,T}^{u_{k,j}}\right]$ into a fixed-size feature matrix $\mathbf{M}^{u_{k,j}}$ of dimensions $\mathrm{N}^{u_k} \times \mathrm{V}^{u_k}$, where V^{u_k} is the dimension of the last hidden layer. We map each element of $\mathbf{M}^{u_{k,j}}$ to the [0,1] space by sigmoid function. Finally, we obtain the fixed-size feature matrix $\mathbf{M}^{u_{k,j}}$ of dimensions $\mathrm{N}^{u_k} \times \mathrm{V}^{u_k}$.

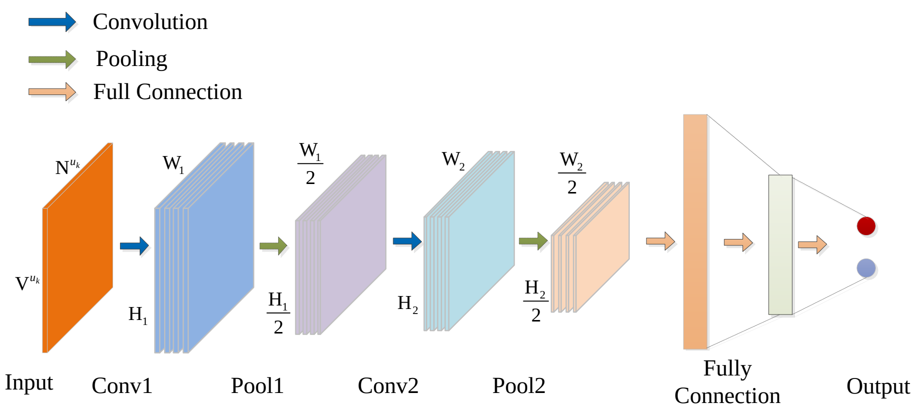


Fig. 3. Structure of the CNN

3.4 Training CNN for Detecting Insider Threat

The final component of our approach is the classification stage. We use the CNN to classify the fixed-size feature matrices of user behavior into normal behavior and anomalous behavior. The CNN consists of an input layer, two convolution-pooling layers, a fully-connected layer, and an output layer. For user u_k, the dimension of the input layer is $\mathrm{N}^{u_k} \times \mathrm{V}^{u_k}$ and the dimension of the output layer is two. Fig. 3 shows the structure of the CNN.

We first train the CNN by using fixed-size feature matrices annotated with normal or anomaly. Also the softmax function is applied to the output of the CNN. After training, we use the trained CNN to calculate anomalous probability of a user action sequence.

Table 1. Enumeration of User Actions

Time	Computer	Activities	ID	Action	Description
In-hourAction (8am and 5pm) or After-hourAction (5pm and 8am)	On an assigned PC or an unassigned PC	Logon/Logoff activity	1	Logon	User logged on a computer
			2	Logoff	User logged on a computer
		File activity	3	Copy exe file	A exe file copy to a removable media device
			4	Copy doc file	A doc file copy to a removable media device
			5	Copy pdf file	A pdf file copy to a removable media device
			6	Copy txt file	A txt file copy to a removable media device
			7	Copy jpg file	A jpg file copy to a removable media device
			8	Copy zip file	A zip file copy to a removable media device
		HTTP activity	9	Neutral website	User visited a neutral website
			10	Hacktivist website	User visited a hacktivist website
			11	CloudStorage website	User visited a cloudstorage website
			12	JobHunting website	User visited a jobhunting website
		Email activity	13	Internal email	All recipients are company email addresses
			14	External email	There is an external address
		Device activity	15	Connect	User inserted a removable media device
			16	Disconnect	User removed a removable media device

4 Experiments

This section reports the experimental validation of the proposed method. We apply our method to the CMU-CERT insider threat dataset [13], which provides a synthetic dataset describing a user's computer based activity. The dataset consists of information on several different activities over a period of 17 months. Next, we first describe details of the dataset and evaluation method. Then we present the experimental results of our approach.

4.1 Dataset

We perform experiments on the CERT insider threat dataset V4.2, because it contains more instances of insider threats compared to the other version of datasets. The dataset captures the 17 months of activity logs of the 1000 users (with 70 insiders) in an organization, which consists of five different types of activities: logon/logoff, email, device, file and http. Each log line is parsed to obtain details like a timestamp, user ID, PC ID, action details etc. We choose a comprehensive set of 64 actions over the five types of activities and build 1000 user specific profiles based on user action sequences. An example of a user action is visiting a job-hunting website between the hours of 8:00 am and 5:00 pm on an assigned computer. The enumeration of user actions is listed in Table 1.

Over the course of 17 months, 1000 users generate 32,770,227 log lines. Among these are 7323 anomalous activity instances manually injected by domain expert, representing three insider threat scenarios taking place.

We split the dataset into two subsets: training and testing. The former subset (~70% of the data) is used for model selection and hyper-parameter tuning. The latter subset (~30% of the data) is used for evaluating the performance of the model. Our classifications are made at the granularity of user-day. One note is that we removed the weekends of the data when we classify at the granularity of user-day, because the user behavior is qualitatively different for weekdays and weekends.

Table 2. Parameters of the LSTM

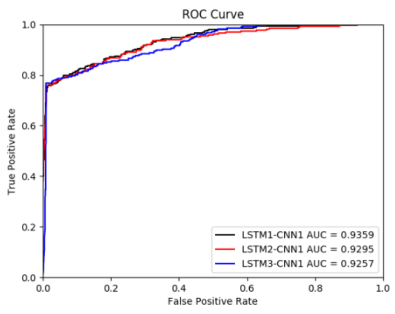
Model	Dimension of three hidden layers	Mini-batch size	Epoch num
LSTM1	60	20	10
LSTM2	40	20	10
LSTM3	20	20	10

Table 3. Parameters of the CNN

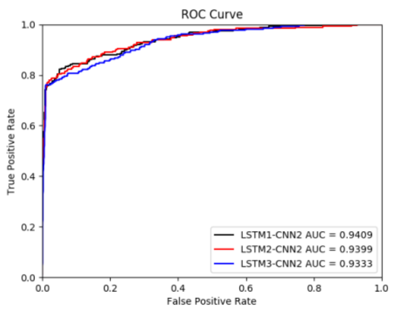
Model	Conv1	Conv2	Activate function	Mini-batch size	Epoch num
CNN1	32(4)	64(4)	tanh	20	500
CNN2	32(5)	64(5)	tanh	20	500
CNN3	32(6)	64(6)	tanh	20	500
CNN4	32(4)	64(4)	relu	20	500

4.2 Evaluation Method

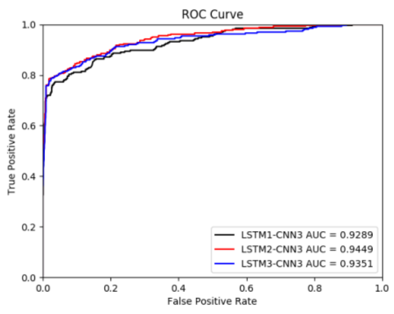
The dataset used for experiment is unbalanced, so we choose the Receiver Operating Characteristics Curves (ROC) and Area-Under-Curve (AUC) measure for evaluating the proposed method. On one hand, we can visualize the relation between TPR and FPR of a classifier. On the other hand, the accuracy with two or more classifiers can be compared.



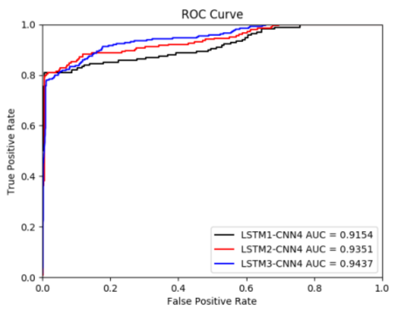
(a) ROC curves for CNN1



(b) ROC curves for CNN2



(c) ROC curves for CNN3



(d) ROC curves for CNN4

Fig. 4. ROC curves for CNNs

4.3 Results

To compare the performance of the model with different parameters, we train our model with several parameters. When setting the parameters of the LSTM, we refer the setting of [14] which uses the LSTM in language modeling. In addition, the LSTM is trained using the ADAM [15] variant of gradient descent. The parameter settings of the LSTM are shown in Table 2.

The parameters of the CNN were set by referring the setting of LeNet [16], which is used for recognizing hand written digit. Let a(b) denotes the number of filters (the shape of each filter) per convolutional layer. Max-pooling reduces the size of the input into 1/2 with stride of 2. The parameter settings of the CNN are shown in Table 3.

We evaluated the ROC curves for each of these CNNs, and later we compare the best performing CNN against the logistic regression classifier-based architectures (see Fig. 5). Fig. 4(a), Fig. 4(b), Fig. 4(c) and Fig. 4(d) show the ROC curves when CNN1, CNN2, CNN3 and CNN4, respectively, are used for classification. We can see that the different parameter settings differ only slightly. The performance of relu activation function is similar to the tanh activation function, using the same parameter settings. The LSTM2 with CNN3 provides better result than the other CNNs and gets the best result AUC = 0.9449.

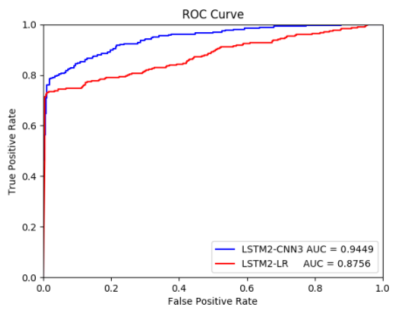


Fig. 5. ROC curves for CNN3 and Logistic Regression

Fig. 5 compares the ROC curves of the best performing CNN3 plus the logistic regression classifier-based architectures. The ROC results for the CNN classifier based architectures are better than the Logistic Regression version with the same language model (LSTM2).

5 Conclusion

In this paper, we proposed the insider threat detection method with deep neural network. Because insider threat manifest in various forms, it is not practical to explicitly model it. We frame insider threat detection as an anomaly detection task and use anomalous behavior of a user as indicative of insider threat. The LSTM extracts user behavior features from sequences of user actions and generates fixed-size feature matrices. The CNN classifies fixed-size feature matrices as normal or anomaly. We evaluated the proposed method using the CERT Insider Threat dataset V4.2. Experimental results show that our method can successfully detect insider threat and we obtained AUC = 0.9449 in best case.

Acknowledgement

This work was partly supported by the National Key R&D Program of China under Grant No.2016YFB0800300, Xinjiang Uygur Autonomous Region Science and Technology Project under Grant No.2016A030007-4, the National Natural Science Foundation of China under grant No. 61602466.

References

1. Gavai, G., Sricharan, K., Gunning, D., Hanley, J., Singhal, M., & Rolleston, R.: Supervised and Unsupervised methods to detect Insider Threat from Enterprise Social and Online Activity Data. JoWUA, 6(4), 47-63(2015).
2. Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., & Robinson, S.: Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. arXiv preprint arXiv:1710.00811(2017).
3. Chandola, V., Banerjee, A., & Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58(2009).
4. B. D. Davison and H. Hirsh.: Predicting sequences of user actions. AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis, pp. 5–12 (1998).
5. T. Lane and C. E. Brodley.: Sequence matching and learning in anomaly detection for computer security. In AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, pp.43–49(1997).
6. R. A. Maxion and T. N. Townsend.: Masquerade detection using truncated command lines. In DSN ’02 Proceedings of the 2002 International Conference on Dependable Systems and Networks, pp. 219–228(2002).
7. Oka, M., Oyama, Y., & Kato, K.: Eigen co-occurrence matrix method for masquerade detection, In Publications of the Japan Society for Software Science and Technology(2004).
8. Szymanski B K, Zhang Y.: Recursive Data Mining for Masquerade Detection and Author Identification. Information Assurance Workshop, pp. 424-431(2004).
9. Rashid, T., Agrafiotis, I., & Nurse, J. R.: A New Take on Detecting Insider Threats: Exploring the use of Hidden Markov Models. In Proceedings of the 2016 International Workshop on Managing Insider Security Threats, pp. 47-56(2016).
10. Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M.: Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In Wireless Networks and Mobile Communications (WINCOM), pp. 258-263(2016).
11. Veeramachaneni, K., Arnaldo, I., Korrapati, V., Bassias, C., & Li, K.: AI2: Training a big data machine to defend. In Big Data Security on Cloud, IEEE International Conference on HPSC, and IEEE International Conference on IDS, pp. 49-54(2016).
12. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580(2012).
13. Glasser, J., & Lindauer, B.: Bridging the gap: A pragmatic approach to generating insider threat data. In Security and Privacy Workshops (SPW), pp. 98-104(2013).
14. Zaremba, W., Sutskever, I., & Vinyals, O.: Recurrent neural network regularization. arXiv preprint arXiv:1409.2329(2014).
15. Kingma, D., & Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980(2014).
16. Theano Development Team, “Convolutional Neural Networks(LeNet)”, http://deeplearning.net/tutorial/lenet.html.
17. Maxion R A, Townsend T N, Masquerade Detection Using Truncated Command Lines. International Conference on Dependable Systems and Networks, pp. 219-228(2002).
18. Maxion R A, Townsend T N, Masquerade Detection Augmented with Error Analysis. IEEE Transactions on Reliability, 53(1), 124-147(2004).
19. Salem, M. B., Hershkop, S., & Stolfo, S. J.: A survey of insider attack detection research. Insider Attack and Cyber Security, pp. 69-90(2008).