Deep Learning Based Anomaly Detection Scheme in Software-Defined Networking

Yang Qin*, Junjie Wei, Weihong Yang
Department of Computer Science,
Harbin Institute of Technology (Shenzhen),
Shenzhen, China,
*corresponding author: csyqin@hit.edu.cn

***Abstract*—Software Defined Networking (SDN) has attracted more and more attention due to its prominent features that are different from the traditional network. SDN is programmable through which controller can modify the rules in the switch. However, security was not considered in its initial design, and many manufacturers no longer support Transport Layer Security (TLS) due to the cost. Although many machine learning based approaches have been implemented in SDN, they all need features that experts extract from original data. However, the manual extraction increases the level of human interaction and decreases detection accurate. This paper presents a malicious network traffic classification method based on Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to address these concerns. Our proposed method is implemented in Graphic Process Unit (GPU) enabled TensorFlow. We evaluated our proposal on three datasets. The results demonstrate that our proposal achieves improvements in term of detection accuracy and stability over existing approaches and strong potential for user in SDN security.**

Keywords—SDN; CNN; RNN; anomaly detection

I. Introduction

In recent years, with the emergence and development of Software Defined Network (SDN), the network structure is more flexible and network management is more convenient [1]. SDN breaks the network function into control logic and forwarding logic: switch becomes a simple forwarding-processing device, and the control logic can be implemented by a central controller. However, some potential security risks have been introduced because security issues are not considered in the initial design. For example, Shin and Gu points out that the Openflow protocol is at risk of being attacked by DDoS (Distributed Denial of Service) [2]. Due to the special logic structure of SDN, once the controller is breached, most of network devices or even the whole network cannot work normally. There are mainly three types of attacks on controllers: fraud attack (e.g., DoS), intrusion attack such as unauthorized login, and malicious tampering attack (e.g., tampering with data in switches). In this paper, we focus on dealing with the first two types of attacks.

There are some recent works using machine learning methods for malicious detection [3-6]. Mehdi et al. [3] proposes to use a programmable router as a platform to detect malicious data streams in an office or home network. Jin and Wang [4] design a system that uses the structure of SDN to analyze real-time data streams to detect malicious flows. Gao et al. [5] uses the deep belief network and NSL-KDD data set for training. Xu and Liu [6] claim that if a host or server is under DDoS attack, they will receive more data streams than the outgoing data stream. However, these methods have obvious deficiencies such as the need for human experts to interact and define data features and models, and low detection accuracy. In order to solve the shortcomings of the above methods, this paper proposes a model for automatically extracting the features of data packets based on CNN and RNN, improving the accuracy, and reducing the level of human interaction.

The main contributions of this paper are summarized as follows. We firstly build a model based on CNN and RNN, which can automatically extract features from original and then classify the malicious network flow in SDN environment. We simulate three common attacks with mininet and pox, using python. We implement the experiment to verify the effectiveness of the model.

The remainder of this paper is organized as follows. Section II covers some related research work, Section III introduces the specific structure of the model, Section IV shows the corresponding experimental results and analysis. Section V concludes this paper.

II. Background

Intrusion detection are generally based on signature and anomaly detection. Signature based method can achieve high detection accuracy of some known attack types. However, the unknown attack cannot be detected. Anomaly-based detection methods are generally solved by machine learning. The features of the network flows are first designed, and then a machine learning algorithm is used to identify new attack network flows [7]. It has the advantage of recognizing the types of attacks currently known and also detecting unknown types of attacks. However, the design of network flow features is a very challenging task. Different attack types may depend on different features. Moreover, the machine learning based detection method has a low detection rate.

Niyaz and Sun [8] utilizes stack auto-encoder to reduce data features, and the proposed algorithm can improve accuracy and reduce computation. An asymmetric stack auto-encoder is proposed in [9] to learn features. Convolutional neural networks can extract data features from layer-by-layer convolution functions and widely used in image classification. Similar to other neural networks, feature extraction is an

important aspect [10]. The features learned in the shallow layer will be input to the next layer to extract more abstract features. A pooling layer between every two adjacent convolutional layers. The role of the pooling layer is to reduce the size of the representation data. The Recurrent Neural Network (RNN) is also consists of an input layer, a hidden layer, and an output layer. But the biggest difference in recurrent neural networks is that, in addition to the connection between layers, there is also a connection between neurons in each layer. However, as the time series is long, the gradient disappears [11]. Therefore, LSTM is proposed. The structure of the cell is retained or changed by the structure of the gates, including input gates, output gates, and forgetting gates [12]. Gated Recurrent Unit (GRU) is a variant of LSTM. GRU keeps the LSTM effect while making the structure simpler. It has only two gates, including the update gate and the reset gate [13].

III. CNN-RNN BASED MODEL

We present the design of the CNN-RNN based model. First, the input data are processed by the preprocess procedure, followed by the algorithm based on CNN and RNN. Finally, the classification is done by linear classifier. The process of the CNN-RNN based model is shown in Fig. 1.

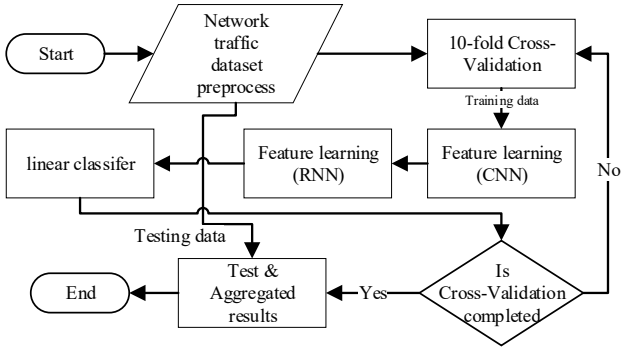

Fig. 1. The CNN-RNN based classification model.

A. Data Processing

We use the pcap to capture network traffic. The original pcap packet contains a 24-byte pcap packet header, a 16-byte data packet header and a data packet. The pcap packet header (as shown in Fig. 2) includes the timestamp of capture, the length of the saved packet, and the true length of the packet. If the packet is fragmented, it is possible that the value of Len is greater than the value of caplen.

Timestamp	caplen	Len

Fig. 2. The format of pcap packet header.

We extract all the contents from packet header. If it is a TCP packet, it will be processed according to the TCP header format of Fig. 3. All the character descriptors in the packets are counted, and then a description dictionary is created. Moreover, every descriptor in the dictionary corresponds to a unique index.

80 1042 SYN ACK Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1

Fig. 3. TCP header.

B. Feature Extraction Based on CNN and RNN

This part mainly includes embedded layer, convolution network and recurrent neural network layer. The main function of the embedding layer is to encode the packet headers into a two-dimensional matrix. The descriptor vector can be pre-trained, or trained in the process of training CNN, but the former one will be 100 times faster. If the pre-trained descriptor vector is used, it is divided into a static method and a non-static method. The former refers to the parameter that no longer adjusts the descriptor vector in the process of training CNN, and the latter adjusts the parameters of the descriptor vector during the training process. Therefore, the latter result is better than the former. Here we pre-train descriptor vectors. The descriptor vectors are shown in Fig. 4.

0.13003991	0.22881057	...	0.24961747	0.08153807
-0.06620334	-0.1124947	...	0.18774514	-0.07204840
-0.06610855	0.00927361	...	0.09578553	0.00616110
-0.09615227	-0.02547764	...	-0.02810406	-0.08124200

Fig. 4. The table of descriptor vectors.

The convolution layer extracts different n-gram features by convolution. The input **A** will be transformed into a two-dimensional matrix **x** by the embedding layer. If the length of the input is $|T|$ and the size of the descriptor vector is $|d|$, the size of the two-dimensional matrix **x** is $|T|\times|d|$. The size of the convolution kernel is generally set to $n\times|d|$, where n is the length of convolution kernel, and $|d|$ is the width of convolution kernel. This width is the same as the dimension of the descriptor vector, that is, the convolution is only along the sequence of inputs. The choices of n can be various, such as 3, 4, and 5. For an input of size $|T|\times|d|$, if the size of the convolution kernel is chosen to be $3\times|d|$, the result obtained after convolution is a vector of $(|T|-3+1)\times 1$. CNN can use multiple different types of kernels at the same time, and there can be more than one kernel for each size. If we use kernels with size of $3\times|d|$, $4\times|d|$, $5\times|d|$ and m kernels per type, the convolutional network has a total of $3\times m$ convolution kernels. For simplicity, we give a simple example in Fig. 5 in which $|T|$ is 7 and $|d|$ is 3.

The max-pooling layer takes the maximum value of several one-dimensional vectors obtained after convolution, and then splices it into one piece as the output value of this layer. If the size of the convolution kernel is 3, 4, and 5, and each size has M kernels, then after max-pooling, $M\times 3$ scalar values will be obtained. Splicing scalar values together, we get the final $(M\times 3)\times 1$ vector. The significance of the max-pooling layer is to extract the most activated features of the convolution extraction.

The RNN layer uses a hidden layer unit of 300 neurons. In this paper, the Gated Recurrent Unit (GRU) core is used. The GRU only contains two control gates: the reset gate and the update gate. The reset gate combines the input data with the network data, and the update gate determines how much of the previous information is retained. If the reset gate in the network is all 1 and the update gate is all 0, then the GRU is a normal RNN network. The reset gate is shown as follow.

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{1}$$

where x_t represents the input at time t which is obtained from CNN model, W_r corresponds to the weight of the input, U_r represents the weight of the previous moment state, and h_{t-1} represents the state of the previous moment state.
The input gate is shown as follow.

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \quad (2)$$

where x_t represents the input at time t which is obtained from CNN model, W_z corresponding to the input weight, U_z represents the weight of the previous time state, h_{t-1} representing the state of the previous moment state. Finally, output candidate $\tilde{h}_t$ and output h_t are computed.

$$\tilde{h}_t = \tanh(W_h x_t + U_t r_t \bullet h_{t-1}) \quad (3)$$

$$h_t = (1 - z_t) h_{t-1} + z_t \tilde{h}_t \quad (4)$$

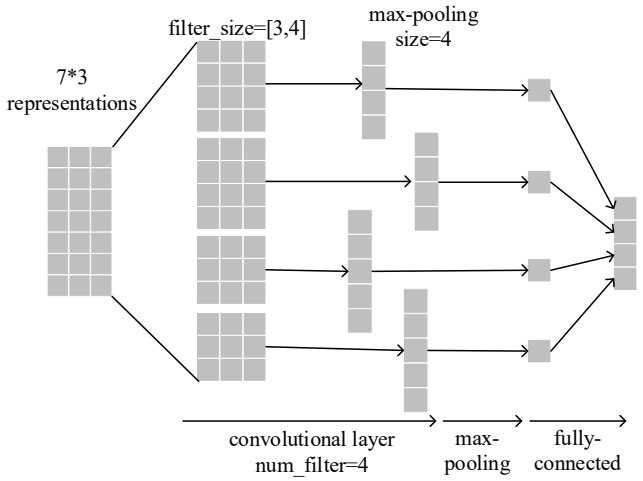

Fig. 5. The process of convolution.

C. Classify

The last layer is linear classifier, as is shown in following formula.

$$f(x_i, w, b) = w x_i + b \quad (5)$$

where x_i represents the input at time t which is obtained from RNN model, w corresponding to the input weight, b represents the bias. In order to evaluate the model, the cross entropy loss function is selected. The specific formula is as follows:

$$loss(Y, \tilde{Y}) = -\sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \times \log \tilde{y}_{ij} \quad (6)$$

where y_{ij} represents the true probability that the i-th sample is classified as j-th class, $\tilde{y}_{ij}$ represent the predict probability that the i-th sample is classified as j-th class. N represent the total number of samples. M represent the total number of classes.

IV. Experiment

A. Metrics

Three metrics were used to evaluate the accuracy of the model: accuracy, recall and F1-score [14]. Accuracy is used to measure the detection performance of the model. The recall rate is used to measure the ability of the model to detect malicious data streams. F1 score is the harmonic average of model accuracy and recall. If F1-score is higher, the model is more stable.

Three metrics were used to evaluate the accuracy of the model. They are accuracy, recall rate and F1-score, respectively [14]. The confusion matrix is shown in Table I. Accuracy is used to measure the detection performance of the model, it indicates the percentage of true detection over total traffic. The recall is used to measure the ability of the model to detect malicious data streams, it shows the percentage predicted intrusions over predicted intrusions and normal traffic traces. F1 score is the harmonic average of model accuracy and recall. If F1-score is higher, the model is more stable.

TABLE I. Confusion Matrix.

True situation	**Predicted result**	
	Positive example	***Negative example***
Positive example	TP	FN
Negative example	FP	TN

B. Dataset

We first generate some network traffic with mininet and pox, which are famous simulators for SDN. Our topology contains one switch and two host. One is attacker and the other is victim. We use python to generate packets in attacker, and then send them to attacker, and finally capture the packets between victim and attacker. We simulate syn flood, port scan, guess ftp password. The self-generated dataset is named sim_data. We compare with a deep learning-based model propose for network intrusion detection in SDN. Besides sim_data, the experiment uses CTU-13 data set [15]. The CTU-13 dataset is the data traffic captured by Stratosphere Labs in real-world environments, including normal traffic, botnet traffic, background traffic. We compare with a deep learning-based malicious flow detection (TSDNN) [16].

C. Settings

The parameters of the convolutional neural network and deep recurrent neural network are shown in Table II. This includes some parameter settings for CNN, such as the size of the convolution kernel, the number of convolution kernels, and the number of training samples. It also includes the parameter settings of the RNN, including the number of hidden layer units, and regularization method that is used to improve the generalization ability of the network.

TABLE II. Experiment Settings.

parameter	**value**
Dropout_keep_prob	0.5
embedding_dim	300
evaluate_every	200
filter_sizes	3,4,5
hidden_units	300
l2_reg_lambda	0.0
max_pool_size	4
non_static	‘false’
num_epochs	1
num_filters	32

D. Results and Analysis

We use 10 cross-validation and set the bach_size at 128 on CTU-13 and sim_data. The model’s best accuracy can reach 99.86% on CTU-13 and 99.84% on sim_data, as shown in the Table III. At the beginning of the calculation, the loss reduces

slowly. As the parameters are updated, the accuracy becomes higher. Because the parameters have big difference from real values. After the data volume reaches a certain level, the parameter updating is no longer obvious, and the recognition accuracy is not improved much. At the same step, CNN-RNN performs better on CTU-13 than sim_data. The main reason is that parameters are updated to be closer to real value on sim_data than CTU-13.

TABLE III. TRAINING PROCESS.

Step	CTU-13		sim_data	
	ACC	LOSS	ACC	LOSS
400	0.9963	0.1844	0.9959	0.004
600	0.9975	0.2283	0.9979	0.0038
800	0.9980	0.0001	0.9980	0.0007
1000	0.9986	0.0008	0.9984	6.931e-05

The relationship between the accuracy of the models and batch_size in Tabel IV shows that as the batch sizes increases, the accuracies of two models decrease. But if the batch sizes is too small, the training process will take more time. Table IV also show in the same batch_size, our CNN-RNN's accuracy is higher than DNN model on both CTU-13 and sim_data. And accuracies of both models are better on CTU-13 than sim_data. The main reason is that our sim_data is obtained in simulating environments.

TABLE IV. ACCURACY WITH DIFFERENT BATCH SIZE.

Batch_size	ACC			
	CTU-13		sim_data	
	CNN-RNN	*TSDNN*	*CNN-RNN*	*TSDNN*
32	0.9989	0.9922	0.9987	0.8501
64	0.9982	0.9922	0.9992	0.8501
128	0.9981	0.9920	0.9984	0.8521
256	0.9972	0.9920	0.9977	0.7703

The CNN-RNN model has a higher recall rate for normal data and F1-score in Table V both on CTU-13 and sim_data. That is our model can detect more normal data network traffic and more stable on CTU-13.

TABLE V. DNN VS CNN-RNN.

Dataset	RECALL		F1-score	
	TSDNN	*CNN-RNN*	*TSDNN*	*CNN-RNN*
CTU-13	0.998	0.9976	0.949	0.9976
sim_data	0.50	0.80	0.33	0.79

V. CONCLUSION

In this paper, we discusse the problems of SDN security and the current approaches to deal with them. We designs a model based on CNN and RNN to learn features without manual interaction. Further, we classify the network traffic using linear classification. We implement our proposed model in TensorFlow. We evaluate our proposal on three datasets (i.e., CTU-13 , CSIC and sim_data). Experiments show that our model can achieve high accuracy, recall and F1 score.

Although our model has achieved the above promising results, we acknowledge that it is not perfect and there is further room for improvement. In future work, the first avenue of exploration for improvement will be to use this model in SDN environment. We will then look to expand upon our existing evaluations by utilizing real-world backbone network traffic to demonstrate the merits of the extended model.

ACKNOWLEDGMENT

This work was supported by the Science and Technology Fundament Research Fund of Shenzhen under grant JCYJ20160318095218091, JCYJ20170307151807788.

REFERENCES

[1] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1. pp. 623–654, 2016.

[2] S. Shin and G. Gu, "Attacking software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, 2013.

[3] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.

[4] R. Jin and B. Wang, "Malware detection for mobile devices using software-defined networking," in *Proceedings - 2013 2nd GENI Research and Educational Experiment Workshop, GREE 2013*, 2013.

[5] N. Gao, L. Gao, Q. Gao, and H. Wang, "An Intrusion Detection Model Based on Deep Belief Networks," in *Proceedings - 2014 2nd International Conference on Advanced Cloud and Big Data, CBD 2014*, 2015.

[6] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *Proceedings - IEEE INFOCOM*, 2016.

[7] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1595–1598.

[8] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based DDoS detection system in software-defined networking (SDN)," *arXiv Prepr. arXiv1611.07400*, 2016.

[9] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Trans. Emerg. Top. Comput. Intell.*, 2018.

[10] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers and Security*. 2017.

[11] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network Traffic Anomaly Detection Using Recurrent Neural Networks," *arXiv Prepr. arXiv1803.10769*, 2018.

[12] L. Bontemps, V. L. Cao, J. McDermott, and N. A. Le-Khac, "Collective anomaly detection based on long short-term memory recurrent neural networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[13] A. F. Agarap, "A Neural Network Architecture Combining Gated Recurrent Unit (GRU) and Support Vector Machine (SVM) for Intrusion Detection," in *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, 2018, pp. 26–30.

[14] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," in *Proceedings - 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM 2016: Green Communications and Networking*, 2016.

[15] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Secur.*, 2014.

[16] Y. C. Chen, Y. J. Li, A. Tseng, and T. Lin, "Deep learning for malicious flow detection," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 2018.