Flow-based Malware Detection Using Convolutional Neural Network

[1]M. Yeo, [1]Y. Koo, [1]Y. Yoon, [1]T. Hwang, [1]J. Ryu, [2]J. Song, *[1]C. Park
[1]Department of Computer Engineering, Kwangwoon University, Seoul, Korea
[2]Department of Computer and Information Security, Sejong University, Seoul, Korea
minsooyeo119112@gmail.com, kuys21@naver.com, yyd0731@gmail.com
, on91716@naver.com, jwryu1128@gmail.com, jssong@sejong.ac.kr, parkcheolsoo@kw.ac.kr

***Abstract*—In this paper, we suggest an automated malware detection method using convolutional neural network (CNN) and other machine learning algorithms. Lately malware detection methods have been dependent on the selected packet field of applications such as the port number and protocols, which is why those methods are vulnerable to malwares with unpredictable port numbers and protocols. The proposed method provides more robust and accurate malware detection, since it uses 35 different features extracted from packet flow, instead of the port numbers and protocols. Stratosphere IPS project data were used for evaluation, in which nine different public malware packets and normal state packets in an uninfected environment were converted to flow data with Netmate, and the 35-features were extracted from the flow data. CNN, multi-layer perceptron (MLP), support vector machine (SVM), and random forest (RF) were applied for classification, which showed >85% accuracy, precision and recall for all classes using CNN and RF.**

Keywords—malware detection; machine learning; deep learning; traffic calssification; flow-based malware classification

Introduction

Recently, as the Internet has been developed globally and dramatically, malware detection has become important. Unlike the past, the latest malwares have various port numbers, protocols and complex structures so that it is difficult to detect the malware [1]. Although there are traditional malware detection methods such as port-based methods and deep packet inspection (DPI) [2], these port-based methods identify traffic associated with specific port from the headers of packets, however, this method likely to lead to uncertain estimation of packets from different port numbers and protocols. DPI is another traditional method to detect malware. However, it is inefficient method to analysis of large number of packets captured in real time because it takes too long time to inspect packets. Faster and more accurate classification methods are required to process large amounts of packet data acquired in real time. The proposed methods in this paper using machine learning algorithms are the most accurate methods with large amount of data, since it is possible to collect large volume packet in real time. In addition, these automatic malware detection methods are much faster than DPI to inspect packets, due to its robust models for large amount of packet data. This paper demonstrates suggested methods are efficient to detect malware with robust models using a large volume of malware packets captured from real environment.

*Corresponding author

I. METHOD

A. Packet data

In this study, the public data from Stratosphere IPS project were used to build malware detection models [3]. Nine different malware packet data were downloaded from web server of Stratosphere IPS. These data sets were acquired before and after the infection, respectively and captured on the window XP environment. The six-class (i.e. Neris, rbot, Virut, Murlo, NSIS and a normal state) were performed using four machine learning algorithms. Table 1 shows more details of acquired data.

TABLE I. THE DETAIL OF ACQUIRED PUBLIC DATA, ARBITRARILY SELECTED 2000 DATA POINT IN EACH CLASS WERE CLASSIFIED DUE TO DATA BALANCE PROBLEM

Data number	Malware name	Number of malware flow
1	Neris	40961
2	Neris	20941
3	rbot	53518
4	rbot	5160
5	Virut	1802
6	Murlo	12254
7	Neris	184987
8	NSIS	2143
9	Virut	40003

B. Preprocessing

Netmate is feature extractor can convert capture files of network packet into 35-flow statistic features [4]. A flow is a sequence of packet IP address and port number from a source to a destination [5]. After feature extraction, the features were normalized with z-score techniques. Table 2 shows the details of 35-statistic features.

TABLE II. DESCRIPTIONS OF THE FLOW FEATURES

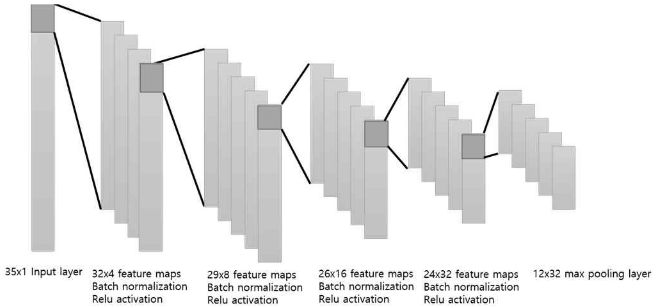
Feature number	Feature name	Feature description
1	**total_fpackets**	Total packets in the forward direction
2	**total_fvolume**	Total bytes in the forward direction
3	**total_bpackets**	Total packets in the backward direction
4	**total_bvolume**	Total bytes in the backward direction
5	**min_fpktl**	The size of the smallest packet sent in the forward direction (in bytes)
6	**mean_fpktl**	The mean size of packets sent in the forward direction (in bytes)
7	**max_fpktl**	The size of the largest packet sent in the forward direction (in bytes)
8	**std_fpktl**	The standard deviation from the mean of the packets sent in the forward direction (in bytes)
9	**min_bpktl**	The size of the smallest packet sent in the backward direction (in bytes)
10	**mean_bpktl**	The mean size of packets sent in the backward direction (in bytes)
11	**max_bpktl**	The size of the largest packet sent in the backward direction (in bytes)
12	**std_bpktl**	The standard deviation from the mean of the packets sent in the backward direction (in bytes)
13	**min_fiat**	The minimum amount of time between two packets sent in the forward direction (in microseconds)
14	**mean_fiat**	The mean amount of time between two packets sent in the forward direction (in microseconds)
15	**max_fiat**	The maximum amount of time between two packets sent in the forward direction (in microseconds)
16	**std_fiat**	The standard deviation from the mean amount of time between two packets sent in the forward direction (in microseconds)
17	**min_biat**	The minimum amount of time between two packets sent in the backward direction (in microseconds)
18	**mean_biat**	The mean amount of time between two packets sent in the backward direction (in microseconds)
19	**max_biat**	The maximum amount of time between two packets sent in the backward direction (in microseconds)
20	**std_biat**	The standard deviation from the mean amount of time between two packets sent in the backward direction (in microseconds)
21	**duration**	The duration of the flow (in microseconds)
22	**min_active**	The minimum amount of time that the flow was active before going idle (in microseconds)
23	**mean_active**	The mean amount of time that the flow was active before going idle (in microseconds)
24	**max_active**	The maximum amount of time that the flow was active before going idle (in microseconds)
25	**std_active**	The standard deviation from the mean amount of time that the flow was active before going idle (in microseconds)
26	**min_idle**	The minimum time a flow was idle before becoming active (in microseconds)
27	**mean_idle**	The mean time a flow was idle before becoming active (in microseconds)
28	**max_idle**	The maximum time a flow was idle before becoming active (in microseconds)
29	**std_idle**	The standard deviation from the mean time a flow was idle before becoming active (in microseconds)
30	**sflow_fpackets**	The average number of packets in a sub flow in the forward direction
31	**sflow_fbytes**	The average number of bytes in a sub flow in the forward direction
32	**sflow_bpackets**	The average number of packets in a sub flow in the backward direction
33	**sflow_bbytes**	The average number of packets in a sub flow in the backward direction
34	**total_fhlen**	The total bytes used for headers in the forward direction
35	**total_bhlen**	The total bytes used for headers in the backward direction.

C. *Classification algorithms*

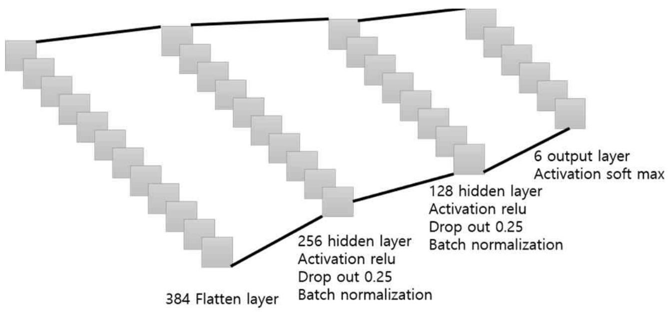
Convolutional neural network (CNN) and three other machine learning algorithms, multi-layer perceptron (MLP), support vector machine (SVM) and random forest (RF), classified six-class flow data with five-fold cross validation. CNN was implemented using Keras packages and other machine learning algorithms were implemented using Scikit-learn packages and python3 programming language was used to build these models. Table 3 shows the parameters of models and figure 1 shows a detail of CNN model structures.

TABLE III. THE PARAMETERS OF MODELS

Algorithm name	Parameter name	Parameter value
MLP	Hidden layer :	5
	Activation	Relu
	Optimizer	Adam
	epoch	200
SVM	C value	1.0
	Kernel function	RBF
RF	Estimators	128
	Max features of trees	16
	Criterion of trees	Gini
CNN	Epoch	50
	Batch size	32
	optimizer	adam



(a)



(b)

Fig. 1. The detailed CNN structures. There are one input layer, five feature map layers, one flatten layer, 2 hidden layers and one output layer. (a) shows a convolutional layer of CNN model and (b) shows a fully connected layer of CNN model.

D. Evaluation

The 3 performance indicators that is accuracy, precision and recall were used to evaluate the performance of the classifiers, which are defined as.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

$$\text{Specificity} = \frac{TN}{TN + FP} \quad (2)$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (3)$$

Where, TP, TN, FP and FN indicate true positive, true negative, false positive and false negative, respectively.

II. RESULT

A. Performance indicators

figure 2 shows the results of six-class malware classification. The x-axis of the figure represents performance indicators that is accuracy, precision and recall and y-axis indicates the values of performance evaluation indicators. All of evaluation indicators of CNN over a 85 percent and the indicators of RF over 93 percent for all classes of the indicators.

B. Confusion matrix

Figure 3 shows the confusion matrix of six-class malware classification. The x-axis of the matrix represents predicted class by the models and y-axis indicates actual class from ground truth.

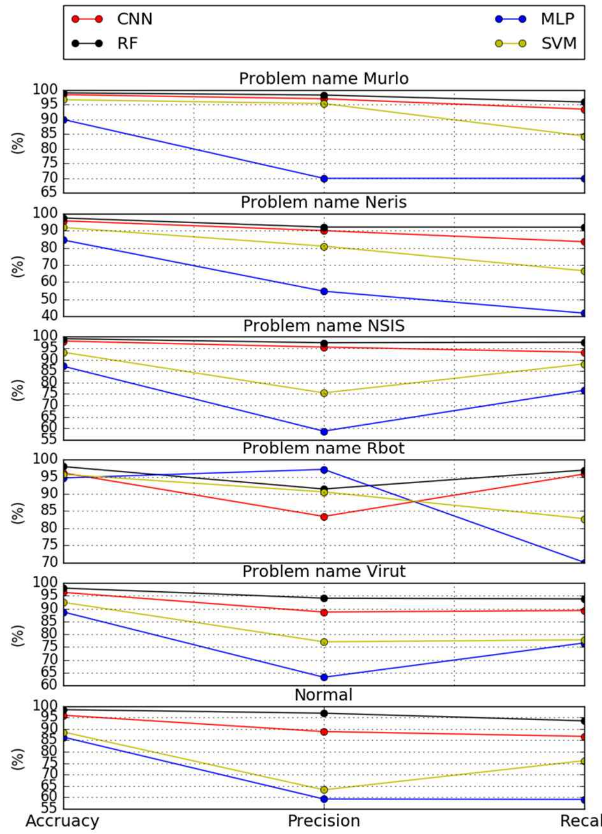


Fig. 2. The result of malware classification. CNN and RF outperform other machine learning algorithms for all performance indicators

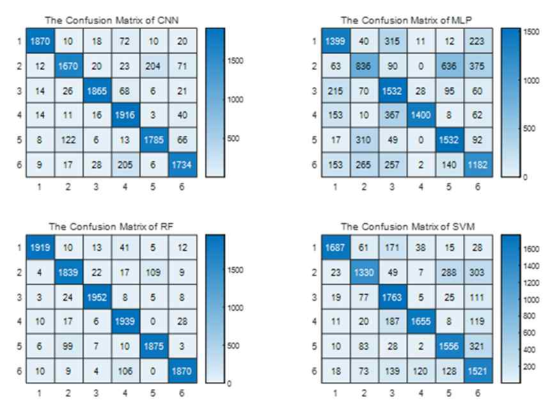


Fig. 3. The details of confusion matrix of 4 machine learning algorithms that is CNN, MLP, RF and SVM. Confusion matrix of RF and CNN lead to superior performance.

CONCLUSION

In this paper, five malware packets and normal state packets were classified with machine learning algorithms. Netmate was used to extract features. CNN and RF achieve superior performance for accuracy, precision and recall.

ACKNOWLEDGMENT

The present research was supported by institute for Information and communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.2016-0-00067, Wise-IoT).

REFERENCES

[1] Moore, Andrew W., and Denis Zuev. “Internet traffic classification using bayesian analysis techniques.” In: ACM SIGMETRICS Performance Evaluation Review. ACM, 2005. pp. 50-60.

[2] Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J. “Deep packet inspection using parallel bloom filters.” In: High performance interconnects, 2003. proceedings. 11th symposium on. IEEE, 2003. pp. 44-51.

[3] Garcia, S., Grill, M., Stiborek, J., Zunino, A., “An empirical comparison of botnet detection methods.” computers & security, vol. 45, 2014. pp. 100-123,

[4] Alshammari, R., and Zincir-Heywood, A. N., “Investigating two different approaches for encrypted traffic classification.” In: Privacy, Security and Trust, 2008. PST'08. Sixth Annual Conference on. IEEE, 2008. pp.156-166.

[5] García, S., Zunino, A., and Campo, M., “Survey on network-based botnet detection methods.” Security and Communication Networks 7.5, 2014. pp.878-903