NETWORK ANOMALY DETECTION

FYP16021 Interim Report

by Tien Hsuan Wu
Supervised by Dr. S. M. Yiu

January, 2017

ABSTRACT

Most anomaly detection systems found in literature are based on data mining methods, and thus are limited to predefined features. The limitations of the existing systems will significantly reduce the performance of the systems if the features are not selected properly. Recently, as the computation power increases, deep learning becomes a popular area of research. In this project, we use deep learning as the model for anomaly detection, and Keras as the library to implement. The model will first be trained with internal network data to classify network packets according to the application layer protocol. The model will then be modified and trained to identify anomalies, and the performance will be evaluated and compared with existing methods. All the training data are internally generated network packets. At this stage, we have studied some deep learning neural network optimization techniques with existing image datasets. We will be focusing on constructing the most effective model for the application layer protocol classification task.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. S. M. Yiu, for motivating my project and guiding me with his immense knowledge.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ANN: artificial neural network

CNN: convolutional neural network

ELU: exponential linear unit

GiB: giga binary byte, 2^{30} bytes

IDS: intrusion detection system

KB: kilobyte in decimal, 1,000 bytes

LSUV: Layer Sequential Unit Variance

MLP: multiple layer perceptron

ReLU: rectified linear unit

TCP: transmission control protocol

I. INTRODUCTION

The Internet is expanding and its scale is increasing as there are more devices that are connected to the Internet. In November, 2016, Google has indexed 46 billion webpages, and the annual global Internet traffic is expected to reach 1 zettabyte (10^{21} bytes) by the end of 2016. With the popularization of the Internet, its usage has become necessary in various areas. However, cyber-attack is a potential threat that comes with this advantage. According to Symantec, there are 54 zero-day (unseen) vulnerabilities discovered each week in 2015, which is twice as many as those in 2014 [1]. Therefore, without appropriate security measures, it is likely that the systems would be compromised, causing individuals and companies suffering from great loss. Intruders may gain unauthorized privileges, or simply overload the server to make it unavailable. Both of these may incur great loss for the system owners.

In order to protect the computers from being hacked, intrusion detection systems (IDS) can be installed. Some common open source IDS are Snort [2] and Suricata [3]. With IDS installed, whenever a system encounters unauthorized access, it can respond by refusing such access request. Moreover, it can generate alerts for human to inspect if there is any system defect.

In our project, we focus on improving the accuracy of the intrusion detection system with deep learning model as backend. The entire model will be built using Keras, and thus we can take advantage of its high modularity to achieve fast prototyping. Currently, have studied optimization techniques with MNIST hand-written digit dataset [4] and CIFAR tiny image dataset [5]. We will move on to focus on determining and building the most effective model. After the model is built, we will train and test it with some internal-generated network data

and evaluate its performance according to metrics such as precision and recall. We will analyze the system and examine the misclassified data to determine the modifications to be made. Finally, we will apply these modifications and reinitiate the training process.

The rest of this interim report is organized as follows. Section II describes the background of intrusion detection system and deep learning. Section III presents the related works to our project. We discuss the methodology in section IV and our schedule and deliverables in section V. The current deep learning optimization research and our experiment results are summarized in section VI. Finally, section VII concludes this report.

II. BACKGROUND

A. INTRUSION DETECTION

Intrusion detection systems can be classified into three categories: signature detection systems, anomaly detection systems, and hybrid systems [6, 7]. A signature detection system maintains a misuse database which contains the patterns of abnormal traffic. When a packet arrives, the system will compare it with the misuse database to determine whether such packet is normal. The advantage is that signature detection systems generate a low false positive rate when the misuse database is reliable. This is due to the fact that intrusions detected are supposed to have a high similarity with the abnormal packets. For an anomaly detection system, it uses the pattern generated from normal traffic as the baseline. Any pattern that deviates from the normal traffic is considered anomalous. The advantage is that it can detect unseen (zero-day) attacks. Hybrid systems combine both techniques used in signature detection systems and anomaly detection systems.

B. DEEP LEARNING

Traditional machine learning methods, such as support vector machines and decision trees, are able to find patterns from a set of data, and so is deep learning. However, what differentiate deep learning with machine learning is the number of learning methods used. In machine learning, typically a single method is used; whereas in deep learning, we can use multiple methods, each method is based on the result of previous one. The deep structure comes from the multiple steps between the input and the output [8].

One advantage of deep learning is its capacity. In traditional machine learning, since only one method is used, it is less desirable to apply it to solve complicated problems such as image recognition and natural language processing. Even if it can fit the training data well through more training iterations (epochs), the model is likely to perform poorly on unseen data. By contrast, the problem can be resolved by setting different classification methods for each step when deep learning is used. With a higher capacity of the model, more types of problems can be solved. We can also expect the model is not overfit so that the accuracy in the training process can truly reflect its performance on unseen data.

In the reality when intrusion detection system is implemented, the IDS needs to accommodate to a wide range of network flow patterns. Moreover, when the size of the deep learning model increases, the model will be able to capture more features and raise the precision of the detection system. Based on these reasons, we chose deep learning framework to perform anomaly detection tasks.

III. RELATED STUDIES

Data mining techniques and machine learning algorithms can be applied to intrusion detection systems, and these techniques have been extensively studied in the past decade. Clustering and classification are some techniques used in IDS. Münz, Li & Carle proposed an anomaly detection system using k-means algorithm which combines both classification and outlier detection [9]. In [10, 11], the authors combined the k-means clustering with naïve Bayes classification. In [12], the authors further utilized the result from k-means clustering as new features for naïve Bayes classifier. In [13], naïve Bayes classifier is combined with decision tree algorithms.

The abovementioned methods operate on network features only, namely, the connection records. To take payload data from packets into consideration, we need some other techniques. PAYL is a histogram-based classification method that takes payload data as input. It builds a histogram from the input, with frequency of each byte pattern being a bin (see Figure 1), and compares the histogram built from the data with baseline [14].

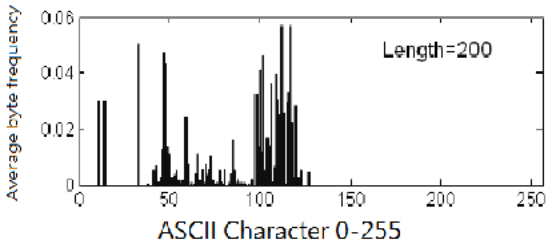

FIGURE 1. Example of byte distribution for a 200-byte packet.

Deep learning techniques can also be implemented to detect anomalies. Niyaz [15] implemented two-stage process of self-taught learning for network anomaly detection. In the first stage, a sparse autoencoder is used for unsupervised feature learning. After the features are learned, they are passed to a softmax classifier in the second stage for anomaly detection. This model can only operate on network features only instead of the entire payload. The performance is evaluated with KDD dataset [16], and precision for 5-class classification is 85.44%. Another deep learning application on network data is studied by Wang [17]. He studied the structures of artificial neural network and stacked autoencoder, and then built a system that can classify TCP packets according to their application layer protocols. The weight coefficients of the first hidden layer can be viewed as the importance of the byte features. Most of the protocols can reach 99% precision in the experiment. He suggested that misclassified packets may be anomalous; however, he did not thoroughly discuss the relationship between those misclassified packets and anomalies. In our project, we will first implement a similar deep learning model and study the misclassified packets. We will then extend such model to detect anomaly packets.

IV. METHODOLOGY

Given a network packet, our first goal is to determine what protocol is used in the application layer. In section II.B, we have discussed processing network packets with deep learning model. As the packets arriving at a host may be scattered, the order and the temporal information will not be useful. Therefore, we can use a deep learning feedforward network as it only captures the features of each single packet. We will use a simple one as an initial implementation, and modify the network to achieve better results.

To implement a deep learning network, it requires some training data to learn from and testing data to evaluate its performance. We will retrieve internal-generated Internet packets from Department of Computer Science, University of Hong Kong. For privacy issues, the source and destination IP addresses of the packets were masked before we start to process them, but the connection information is still preserved. In other words, suppose there were two hosts A, B that sent several packets to each other. We could not know who host A and host B were, but we could still filter out the packets that involved these two hosts.

The next step is to adjust the data so that they are suitable for deep learning. Since the model requires fixed size of input, we had to truncate packets that are too large and pad the packets that are too small. We will set the size of the packets to 500 bytes since previous research showed that most important bytes are located in the first few bytes (see Figure 2) [17]. We will not merely use the first 100 bytes because it contains too limited information, and we will not use the entire 1 KB for the reason that it causes too much overhead in training process.

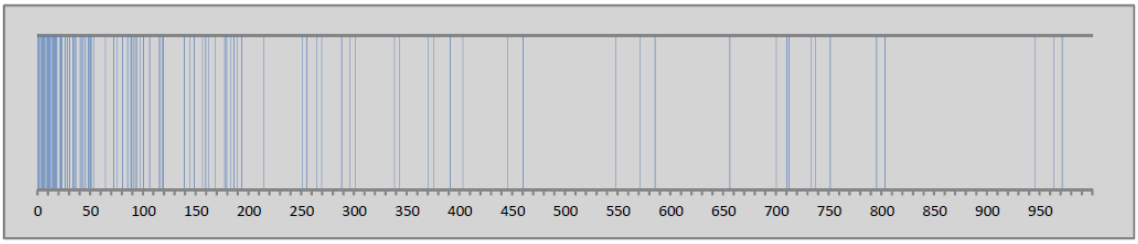

FIGURE 2. The most important 100 locations of the data.

In order to modify the feedforward network, we have to first determine the factors that affect the performance of the system, such as the number of layers of the system and the learning algorithm used. After the neural network is built, we will study the relationship between the byte features and their importance in protocol identification. We will verify the result of the byte features by examining the packets of the neural network. As the baseline (normal behavior) for different applications may vary, the result derived can help in the second phase, that is anomaly detection.

Our ultimate goal is to identify packet outliers. Based on the system we will have built, we would be able to use similar techniques. The difference is that in the training set, we will include some anomalous data, including attack traffic in the Internet. For the output, it should be able to distinguish anomaly packets from normal flow.

We chose Keras for implementing deep learning platform because its highly modularity allows the program development and prototyping in a relative simple manner. In addition, as we focus on evaluating the existing deep learning structures and the effects of parameters, we can exploit the advantages of modularity and avoid extensive changes in source code during training phase. Another concern in development is the compatibility and extensibility with

other packages. In python, which Keras is written in, there are other machine learning packages that can be utilized. Therefore, Keras is particularly suitable for our project.

V. Project Schedule and Deliverables

A. Difficulties and Limitations

Apart from the structure of the deep learning model, one of the key factors that affects the efficiency of the model is the training data. When more training data are collected, the deep learning model can learn from more examples and find the set of patterns that can be applied in general. Therefore, after the model is trained, it will perform well on the training data and unseen test data. The chance for the model to overfit training data is reduced. In addition, if the training data used can reflect the real situation of the Internet, the model will be more applicable to practical use.

In our project, we will use self-generated network packets because we have no access to anomalous packets. Thus, the training data may be insufficient and there is a lack of variety in the anomalous packets.

B. Deliverables

At the end of September, 2016, we delivered a detailed project plan and a webpage (http://i.cs.hku.hk/fyp/2016/fyp16021/) that continuously maintains the progress of our work. We are now collecting network data and building preliminary deep learning model. After the project is completed, source code, the dataset that is used for training and evaluation, and a report that summarizes the performance of our deep learning model will be delivered.

The project schedule is as follows:

Done	• *Study the theory of deep learning* • *Familiarize myself with deep learning model constructions*
	Deliverables of Phase 1 (Inception) • Detailed project plan • Project web page
November-December	*Deep learning optimization*
11-15 January 2017	First presentation
24 January 2017	Deliverables of Phase 2 (Elaboration) • Preliminary implementation • Detailed interim report
February - Mid March	*Development of protocol classification*
Mid March – Mid April	*Development of anomaly detection*
17 April 2017	Deliverables of Phase 3 (Construction) • Finalized tested implementation • Final report
18-22 April 2017	Final presentation
3 May 2017	Project exhibition
6 June 2017	Project competition (for selected projects only)

TABLE 1. Project Schedule

VI. Deep Learning Optimization

For a deep learning neural network to classify data samples correctly, the model is required to have enough capacity and the training data have to represent general situations. In this section, we will discuss some methods to build a model that achieves a better performance. Since there is not much research related to deep learning optimization with network data, we will use the datasets that are widely studied in the literature for experiment.

A. Dataset

MNIST hand-written digit database [4] is one of the widely used labeled data for machine learning. Each grayscale written digit is normalized in size and centered in a 28x28 image. The dataset contains 60,000 training samples and 10,000 test samples for performance evaluation.

Another dataset that is popular in deep learning is CIFAR-100 colored tiny image dataset. Each image has a coarse label and a fine label, which belongs to one of the 20 superclasses and one of the 100 classes (see Table 2). In our experiment, we use the fine labels only. The size of images is 32x32.

FIGURE 3. The MNIST dataset

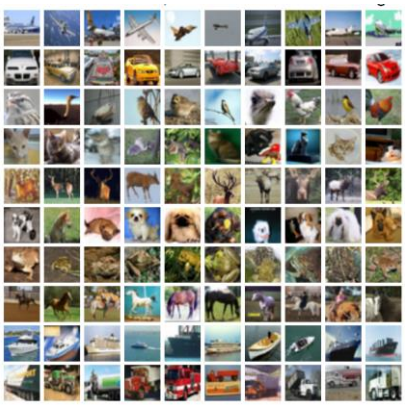

FIGURE 4. The CIFAR-100 dataset

Superclass (Coarse)	**Classes (Fine)**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

TABLE 2. CIFAR-100 labels

B. EXPERIMENTS WITH MLP AND CNN

We first built a multiple layer perceptron (MLP) and convolutional neural network (CNN) according to the configuration in Table 3 to test against MNIST dataset. In this section, all the models were trained on the fyp server (single core Intel i7 CPU, 2GiB memory). Both models are trained with stochastic gradient descent (learning rate = 0.01). The batch size was set to 100 and model was trained for 20 epochs, so there were 12k parameter updates for each model during training. The result is reported in Table 4.

<table>
<tr><th>MLP</th><th>CNN</th></tr>
<tr><td colspan="2">Input 28x28 images</td></tr>
<tr><td>400 units</td><td>30 filters 3x3</td></tr>
<tr><td colspan="2">Activation (ReLU)</td></tr>
<tr><td colspan="2">Dropout (0.2)</td></tr>
<tr><td>400 units</td><td>30 filters 3x3</td></tr>
<tr><td colspan="2">Activation (ReLU)</td></tr>
<tr><td colspan="2">Dropout (0.2)</td></tr>
<tr><td rowspan="2">400 units</td><td>Maxpooling (2x2)</td></tr>
<tr><td>120 units</td></tr>
<tr><td colspan="2">Activation (ReLU)</td></tr>
<tr><td colspan="2">Dropout (0.2)</td></tr>
<tr><td colspan="2">10-way softmax</td></tr>
</table>

TABLE 3. Experiment configuration for MNIST

	MLP	CNN
No. of parameters	638,810	528,160
Test Accuracy	0.9667	**0.9769**
Test Loss	0.1051	0.0783
Train Accuracy	0.9601	0.9719
Train Loss	0.1369	0.0913
Training time	260s	3000s

TABLE 4. Experiment result for MNIST

From Table 4, we can see that convolutional neural network can perform better than a multiple layer perceptron even with fewer number of parameters. The training time for CNN can be improved if we have a GPU and train the model with parallelization.

We also performed a similar experiment on CIFAR-100 dataset. We set different batch size for MLP and CNN: 128 and 32, respectively. After 100 epochs of training, there will be 39k parameter updates for MLP and 156k parameter updates for CNN. As the numbers of parameter updates are different, we also report the performance of CNN after 25 epochs of training (39k parameter updates, similar to that of MLP). The training was done with stochastic gradient descent with learning rate 0.025, decay 10^{-6} and Nesterov momentum 0.9. The configuration is shown in Table 5 and the result is reported in Table 6.

From the result, we can see that as the task becomes more difficult, the difference in parameters and performance for MLP and CNN are greater. One property for CNN is that it captures the relationship of the neighboring area. Thus, in the image classification task, CNN

prevails over MLP in detecting edges and contours, making the entire classification more accurate. However, there is a lot of improvement left in the level of accuracy. In the next subsection, some state-of-the-art research in deep learning optimization will be discussed.

MLP	CNN
Input 32x32 images	
3000 units	32 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	32 filters 3x3
3000 units	Activation (ReLU)
Activation (ReLU)	Maxpooling (2x2)
Dropout (0.2)	Dropout (0.25)
2000 units	64 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	64 filters 3x3
2000 units	Activation (ReLU)
Activation (ReLU)	Maxpooling (2x2)
Dropout (0.2)	Dropout (0.25)
1000 units	512 units
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	Dropout (0.5)
100-way softmax	

TABLE 5. Experiment configuration for CIFAR-100

	MLP	CNN (25 epochs)	CNN (100 epochs)
No. of parameters	30,327,100	1,297,028	
Test Accuracy	0.2535	**0.4002**	**0.4270**
Test Loss	4.1776	2.3355	2.2208
Train Accuracy	0.5805	0.3235	0.3705
Train Loss	1.5860	2.6801	2.4434
Training time	20000s	6750s	27,000s

TABLE 6. Experiment result for CIFAR-100

C. Optimization Research

The best result in CIFAR-100 uses exponential linear units (ELU) [18] in the activation layers. With non-zero mean activation layers, each layer will produce a bias and propagate to the next layer, which makes the gradient descent less efficient for optimization. ELU is claimed to reduce such bias shift effect. The exponential linear unit is: ($\alpha > 0$)

$$f(x) = \begin{cases} x & ,x > 0 \\ \alpha(\exp(x) - 1) & ,x \leq 0 \end{cases}, f'(x) = \begin{cases} 1 & ,x > 0 \\ f(x) + \alpha & ,x \leq 0 \end{cases}$$

Figure 5 from [18] shows the relationship of ELU, leaky ReLU, ReLU and shifted ReLU. The experiment done by Clevert, et al. achieved 75.72% accuracy in CIFAR-100 dataset, and Table 7 shows the model used in their experiment.

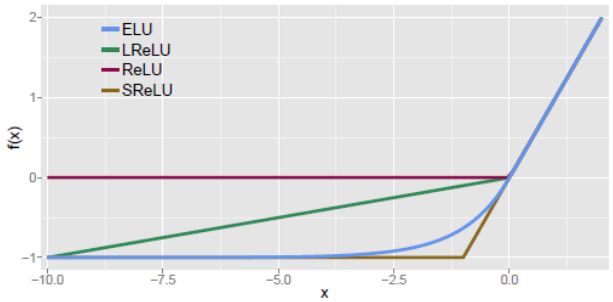

FIGURE 5. Activation functions

Input 32x32 images
384 filters 3x3
ELU (α=1)
Maxpooling 2x2
384 filters 1x1
384 filters 2x2
640 filters 2x2
640 filters 2x2
ELU (α=1)
Dropout(0.1)
Maxpooling 2x2
640 filters 1x1
768 filters 2x2
768 filters 2x2
768 filters 2x2
ELU (α=1)
Dropout(0.2)
Maxpooling 2x2
768 filters 1x1
(continue on the next column)

896 filters 2x2
896 filters 2x2
ELU (α=1)
Dropout(0.3)
Maxpooling 2x2
896 filters 3x3
1024 filters 2x2
ELU (α=1)
Dropout(0.4)
Maxpooling 2x2
1024 filters 1x1
1152 filters 2x2
ELU (α=1)
Dropout(0.5)
Maxpooling 2x2
1152 filters 1x1
ELU (α=1)
100-way Softmax

TABLE 7. Configuration in the original experiment of ELU

Another research in the deep learning optimization focuses on the weight initialization. Initializing the model with Gaussian noise $\mathcal{N}(0, 0.01^2)$ became popular after CNN showed its success in 2012 [19]. Glorot & Benigo [20] proposed a formula to estimate the standard deviation, under the assumption that the relationships between each layer is non-linear. Mishkin & Matas generalized the previous method and named it Layer Sequential Unit Variance (LSUV) [19]. The overall performance on CIFAR-100 is 72.34% accuracy rate.

There are more optimization methods that are developed recently. Adaptive piecewise linear activation unit [21] is able to learn the activation functions. Fractional max-pooling [22] ameliorates the effect on reducing the size during forward propagation. The pooling is not restricted to the fraction of 1/k where k is an integer, and a pooling fraction between 1/2 and 1 has demonstrated an improved performance. Another method related to pooling is the all convolutional net [23]. The pooling layer is replaced by a convolutional layer with an appropriate stride (also called subsampling).

D. Experiments with Advanced Techniques

In this subsection, we focus on the experiment with ELU and LSUV discussed previously. As the model used in [18] is too large and beyond the computation power supported by the server, we used a model with a reduction in size. The model configuration is shown in Table 8.

The training was done with stochastic gradient descent with decay 10^{-6} and Nesterov momentum 0.9. Batch size was set to 100. The learning rate schedule we applied was: 0.005 [1-200 epochs], 0.0025 [201-400 epochs], 0.0005 [401-500 epochs].

Input 32x32 images
80 filters 3x3
80 filters 1x1
ELU (α=1)
Maxpooling 2x2
140 filters 3x3
140 filters 2x2
ELU (α=1)
Dropout(0.1)
Maxpooling 2x2
180 filters 2x2
180 filters 1x1
ELU (α=1)
Dropout(0.2)
Maxpooling 2x2
200 filters 2x2
200 filters 1x1
ELU (α=1)
Dropout(0.3)
Maxpooling 2x2
512 units
ELU (α=1)
Dropout(0.5)
100-way Softmax

TABLE 8. Configuration in our experiment with ELU and LSUV

The training for each epoch took 740 seconds. After 500 epochs of training, the test accuracy reached 70.15% and the training accuracy was 79.15%.

We then used the same model and replaced the ELUs with different activation units. The settings for training were the same, except the schedule for learning rate being: 0.01 [1-100 epochs], 0.001 [101-200 epochs], 0.0001 [201-300 epochs]. The results are compared in Table 9.

	ELU	ReLU	LeakyReLU
Test Accuracy	**0.6837**	0.6523	0.6773
Test Loss	1.1104	1.2186	1.1202
Train Accuracy	0.7076	0.6649	0.6953
Train Loss	0.9757	1.1291	1.1301

TABLE 9. Comparing different activation functions

Rectified linear unit (ReLU) has an activation of 0 when the input is negative; therefore, the mean activation is always non-negative. Both ELU and LeakyReLU have negative activations. Theoretically, ELU performs better than LeakyReLU because the mean of activations is closer to 0. The experiment results are consistent with the theory.

VII. CONCLUSION

As people nowadays heavily rely on the Internet, it is important to develop an intrusion detection system that helps prevent existing as well as zero-day network attacks. With the increase of computer power, it is feasible to construct a deep learning model to detect anomalies based on payload data of packets. In our project, we will first construct a deep learning neural network that is able to classify network packets according to its application layer protocol. The model for detection will be used as a foundation to construct the deep learning anomaly detection system.

REFERENCES

1. Symantec. Internet Security Threat Report 2016 2016. Available from: https://www.symantec.com/security-center/threat-report.
2. Cisco. Snort 2016. Available from: https://www.snort.org/.
3. Open Information Security Foundation. Suricata 2016. Available from: https://suricata-ids.org/.
4. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86:2278-324.
5. Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. 2009.
6. Patcha A, Park J-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer networks. 2007;51:3448-70.
7. Agrawal S, Agrawal J. Survey on Anomaly Detection using Data Mining Techniques. Procedia Computer Science. 2015;60:708-13.
8. Goodfellow I, Bengio Y, Courville A. Deep Learning. 2016.
9. Münz G, Li S, Carle G. Traffic anomaly detection using k-means clustering. GI/ITG Workshop MMBnet; 2007.
10. Chitrakar R, Huang C. Anomaly based Intrusion Detection using Hybrid Learning Approach of combining k-Medoids Clustering and Naïve Bayes Classification. 2012 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM); 2012.
11. Muda Z, Yassin W, Sulaiman M, Udzir NI. A K-Means and Naive Bayes learning approach for better intrusion detection. Information Technology Journal. 2011;10:648-55.
12. Varuna S, Natesan P. An integration of k-means clustering and naïve bayes classifier for Intrusion Detection. 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN); 2015.
13. Farid DM, Harbi N, Rahman MZ. Combining naive bayes and decision tree for adaptive intrusion detection. arXiv preprint arXiv:10054496. 2010.
14. Wang K, Stolfo SJ. Anomalous payload-based network intrusion detection. International Workshop on Recent Advances in Intrusion Detection; 2004.

15. Niyaz Q, Sun W, Javaid AY, Alam M. A Deep Learning Approach for Network Intrusion Detection System. 9th EAI International Conference on Bio-inspired Information and Communications Technologies; 2015.
16. Hettich S, Bay SD. The UCI KDD Archive. Irvine, CA: University of California, Department of Information and Computer Science1999.
17. Wang Z. The Applications of Deep Learning on Traffic Identification. Black Hat; 2015.
18. Clevert D-A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:151107289. 2015.
19. Mishkin D, Matas J. All you need is a good init. arXiv preprint arXiv:151106422. 2015.
20. Glorot X, Bengio Y, Understanding the difficulty of training deep feedforward neural networks. 2010: Publisher.
21. Agostinelli F, Hoffman M, Sadowski P, Baldi P. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:14126830. 2014.
22. Graham B. Fractional max-pooling. arXiv preprint arXiv:14126071. 2014.
23. Springenberg JT, Dosovitskiy A, Brox T, Riedmiller M. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:14126806. 2014.