Unsupervised Approach for Detecting Low Rate Attacks on Network Traffic with Autoencoder

Conference Paper · June 2018

DOI: 10.1109/CyberSecPODS.2018.8560678

3 authors:

Baskoro Adi Pratomo
Cardiff University
19 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Pete Burnap
Cardiff University
88 PUBLICATIONS **1,728** CITATIONS
SEE PROFILE

George Theodorakopoulos
Cardiff University
16 PUBLICATIONS **42** CITATIONS
SEE PROFILE

Unsupervised Approach for Detecting Low Rate Attacks on Network Traffic with Autoencoder

Baskoro Adi Pratomo*
School of Computer Science and Informatics
Cardiff University
Cardiff, UK
Informatics Department
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
*me@baskoroadi.web.id

Pete Burnap†, George Theodorakopoulos‡
School of Computer Science and Informatics
Cardiff University
Cardiff, UK
†BurnapP@cardiff.ac.uk, ‡TheodorakopoulosG@cardiff.ac.uk

***Abstract*—Most approaches to network intrusion detection look only at the header part of network packets. These approaches are able to detect high-rate attacks, such as Denial of Service or probing, with high degrees of accuracy. However, it remains to be seen whether they are also able to detect more subtle attacks, such as when adversaries try to exploit a vulnerability or plant a backdoor. In these cases, the attributes of network packets are usually very similar to the legitimate traffic which presents a limitation for header-only intrusion detection methods. Such attacks present an increasing problem to network security, especially given the rise of Internet of Things (IoT) and the rapidly increasing number of devices that can be exploited through low-intensity attacks. To address this problem we propose the use of the Autoencoder method for network intrusion detection. Autoencoder is a deep learning architecture that has the capability to identify outliers in a dataset. Thus it does not need labelled datasets which contain both legitimate and malicious traffic for training purposes. Through our experiments, we show that the proposed approach was able to detect 100% of low rate attack traffic with an average false positive rate of 8.01%. To demonstrate the improvement over the state of the art we have compared our results to a number of other similar works and our proposed method gave at least 32.81% better in detection rate.**

***Index Terms*—payload inspection, intrusion detection, autoencoder, deep learning.**

I. Introduction

Variation and complexity of attacks on computer networks are increasing rapidly. Anomaly-based Intrusion Detection Systems (IDS) have received significant research attention to develop innovative ways to detect attacks with many of the previous works in this area focusing on the statistical properties of network traffic [5][11]. This is typically achieved by inspecting only the header of packets. Prior research in this area has shown promising results, particularly in detecting Denial of Service (DOS) and probing attacks, in which the attacker usually sends many more packets than would be deemed 'normal'. Nevertheless, such approaches may fail to detect *low rate* attacks that have similar statistical properties to the legitimate traffic but have specially crafted messages. Attacks which contain exploits, backdoors and attacks on Web-based applications (e.g. SQL Injection and Cross Site Scripting) fall into this category. These attacks tend to exhibit low rate traffic because they send only the necessary messages to exploit a vulnerability rather than trying to overwhelm the services.

Therefore this paper focuses on inspecting the content of application layer protocol packets to distinguish legitimate from malicious traffic. This content is referred to as the payload. To do that, there are two options, classification and outlier detection. With classification, first, we need to have a labelled training set that clearly mentions whether the network traffic is malicious or legitimate. Then a model is trained to distinguish them by learning what the differences are. However, this approach has several drawbacks. Firstly, it needs a well-labelled dataset, while in real world it is hard to obtain labelled network traffic that suits every situation. Each institution has different behaviour which affects what kind of data being sent on their network. Thus it is not possible to use a labelled traffic of an institution to train a model in the others. Secondly, the classification approach is prone to error when it faces an imbalanced dataset where the number of samples in each class are heavily different. In our case, the data are most likely to be imbalanced, since the amount of legitimate traffic is always undoubtedly bigger than the malicious one. Thus, its result could be misleading as the method could classify most samples to the bigger class and still get a good accuracy. On the other hand, outlier detection (also known as one-class classification) does not need a labelled dataset to train. It just needs samples of normal behaviour as training set and then can detect anomalies in the new data. It also works well with an imbalance dataset.

As an evidence that outlier detection is more favourable in the area of payload inspection, there are some previous works that employed it. PAYL[22], Poseidon[4], Anagram[21], and RePIDS[9] leverage clustering-based approaches. McPAD[16] and its predecessor[17] utilise a one-class Support Vector Machine (SVM), while OCPAD[20] uses one-class Naive Bayes.

Hence, we propose to use an outlier detection approach as it does not require the training data to be labelled nor balanced

and is capable of learning from samples of a single class. Thus, in this case, we train an Autoencoder by presenting it with samples of legitimate traffic, and we test it against low-rate attacks to determine the effectiveness of detecting those attacks. Autoencoder applies backpropagation to set the target values to be equal to the inputs. It is usually used to learn a representation for a set of data, but in this case, we utilise it to look for outliers by looking at the difference between input and output values, which then be referred as the reconstruction error. Outliers would be spotted when the difference is high.

To accurately measure whether new incoming traffic is highly far from the normal model, the reconstruction error of the incoming traffic is compared to a threshold which is obtained during the training phase. If the error is larger than the threshold, then that traffic is deemed malicious. This approach of using threshold is widely used in the previous works, although they have different approach to calculate the threshold. For instance, PAYL[22] calculates the average distance between the centroids and members of the cluster of the training data. OCPAD[20] uses the average of probability vector of the training data. To calculate the average, they use mean and standard deviation. The problem with using mean and standard deviation is they are not reliable when the data are skewed, whilst that is the case in our approach since the reconstruction errors of the training data is skewed. Thus, we also propose the use of two different methods to statistically calculate the threshold which is more suitable for skewed data.

To benchmark the performance of our proposed system to detect low rate attacks and evidence an improvement over the state of the art, we compare the results of our experiment with other payload-based IDS. These previous works were chosen due to their source code or pseudocode availability and were trained using the most recent dataset, UNSW-NB15[15], which also contains low rate attacks for testing purposes. Therefore, the main contributions of our work include:

- Utilising a deep learning architecture (Autoencoder) as an outlier detector to identify low rate attacks in network traffic and examining the effect of different method to calculate threshold. To the best of our knowledge, nobody has done this. Javaid et al did utilise Autoencoder[11], however, their approach uses statistical attributes of network traffic and runs over the NSL-KDD dataset. Thus, our approaches are completely different.
- Evaluating the proposed method along with other previous works[22], [20] using newer network traffic datasets.
- Discussing recently publicly available network traffic dataset and giving comparisons to the popular DARPA 99 dataset.

The rest of the paper is structured as follows. Section 2 discusses related work on payload-based IDS and identifies current limitations. Section 3 details the datasets which will be used for the experiment. Then our proposed method is explained in detail in Section 4. The results of the experiment are presented in Section 5 and the paper will conclude in Section 6.

II. Related Work

Payload-based IDS have been proposed by several researchers. Mahoney et al. [13] proposed a hybrid IDS that analysed both packet header and payload. They use the occurrence probability of certain keywords in header lines of application layer messages. They only pay attention to the first word from those lines. This could be bypassed because shellcodes usually lie in the body part of application layer message. Wang et al. proposed PAYL [22], which uses all the bytes in a payload to create 1-gram representations of those bytes. It builds a normal model based on the sets of 1-grams and applies a simplified Mahalanobis distance to see how far the new incoming packet is from the model. Another work, Poseidon [4] improved on PAYL by pre-clustering the data using a Self Organizing Map, thus resulting in a smaller number of clusters. RePIDS [10] calculates the distance between 1-grams and stores it in a Mahalanobis Distance Map (MDM). It then reduces the dimensionality of the MDM and uses it as the normal model. The distance between a new packet and the model is then calculated. The distance calculation and decision making of these three methods involve mean and standard deviation. However, mean and standard deviation are sensitive to outliers in training data [6].

Anagram [21] further developed the PAYL method by collecting high order n-grams of bytes from network packets and identifying how many n-grams have not been seen previously. The numbers of newly seen n-grams are used to calculate an anomaly score. Rieck et al. [18] stored the n-gram representation of bytes in a *trie* data structure and experimented with various distance functions to detect anomalies. McPAD [16] and its predecessor [17] implemented multiple Support Vector Machine (SVM) classifiers and modified n-grams of bytes to $2v$-grams that took the first and last characters of a sequence of bytes with length v. HMMPayl [3] is another development on PAYL which utilises Hidden Markov Models to detect anomalies. OCPAD [20] stores the n-grams of bytes in a probability tree and use one-class Naive Bayes classifier to classify malicious traffic.

All the aforementioned research on payload inspection built a normal model of network traffic and then aimed to detect outliers with a threshold that act as a borderline to clearly separate them. However, all of them only process the network packets individually, which means the decision between two IP packets that belong to the same TCP stream could be different. Attacker thus can leverage the absence of TCP reassembly to avoid detection. Moreover, none of them inspected the input to their threshold calculation method to check which distribution their data belong to before using mean an standard deviation as part of their threshold.

Furthermore, some of these works [13], [22], [4], [12] were tested on the DARPA 1999 IDS evaluation dataset, which is now 19 years old. Their results may be different if the experiments were repeated with newer malicious traffic, as attacks are always evolving.

III. Datasets

Due to the critiques of DARPA 99 dataset, Shiravi et al.[19] introduced the concept of profiles in building an IDS dataset called ISCX IDS dataset (ISCX 12). They did this in order to make the dataset reproducible, though the PCAP files of their dataset were also released. The profile works as a guide for everyone who wants to reproduce the dataset, that then the result must be captured by sniffing applications, e.g Wireshark or tcpdump.

ISCX 12 consists of seven days of network traffic from 11 June to 17 June. The only clean, attack-free, dataset is 11 June - all the others contain attack traffic. Most of them are HTTP based DOS, IRC botnet, and SSH brute force attacks. This dataset also has attacks that stem from PDF buffer overflow vulnerability and SQL Injection. Most of the packets, whether they are benign or malicious, are HTTP packets, though it also contains other protocols (i.e. FTP, SMTP, and IMAP) in smaller samples. However, the label of this dataset is not clear, because they only label the traffic as normal or attack. Thus, it is not clear what kind of attack they belong to and makes it hard to explain the result.

Another publicly available dataset that has captured network traffic files is UNSW-NB15[15]. This dataset comprises two days of captured network traffic, the first one was captured on 22 January 2015 and the second traffic was collected on 17 February 2015. The difference between those two days is the amount of traffic since the second days has roughly ten times more traffic. Both of them contain ten classes of traffic, i.e. Normal, Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms. The explanation of those classes is shown on Table I. These attacks are generated based on recent Common Vulnerabilities and Exposures(CVE), thus they are also deemed realistic.

Therefore, since UNSW-NB15 is the most recent publicly network traffic dataset and it has a clearer representation of malicious traffic than ISCX12, then this dataset is used as our benchmark to evaluate the performance of our proposed method and other previous works.

TABLE I
Description of classes in UNSW-NB15 dataset

Class Name	Description
Normal	Legitimate traffic without any malicious payload
Analysis	Port scanning, spam, and HTML file injection
Backdoors	Planting malicious files as an alternative entrance to the server
DoS	Denial of service attack
Exploits	Attacks that exploit vulnerabilities in the server to make it doing something unintended
Fuzzers	Sending various inputs to an application to find vulnerabilities
Generic	Attacks that try to break block ciphers
Reconnais-sance	Used to gather information of the server
Shellcode	Small piece of code which used to run a program on the remote server
Worms	A piece of code that can replicate themselves, sometimes through the network as well

IV. Methodology

This section details our approach to detecting low rate attacks in network traffic. The first part of this section will explain how we preprocess and transform the network traffic from raw captured packets to features for the outlier detector. Afterwards, the second part explains how we trained the outlier detector. We conclude the section by explaining how the proposed system distinguishes legitimate and malicious traffic.

A. Network Traffic Preprocessing

In order to correctly analyse the application layer messages, they need to be reconstructed since an application layer message might be split into several TCP segments. Therefore when a network packet arrives, it is not directly processed by the outlier detector. It is put in a queue buffer instead and to reassembly the TCP segments to an application layer message, we decided to implement it from scratch by following the RFC 793 [2]. Since there is a bug in the PyNIDS[14] Python library for TCP stream reassembly that includes arbitrary bytes in the reconstructed messages and to the best of our knowledge there is no other matured library for reassembling TCP stream. To ensure the correctness of our implementation, we compare the reconstructed application layers messages with the Follow TCP Stream feature in Wireshark and they are matched.

A normal TCP connection is ended when a FIN packet is received, by the time this happens, the connection is marked as ready and can be examined by the outlier detector. However, connections could be disconnected for various reasons, thus we also monitor how long a TCP connection has been in the buffer. If it has been in the buffer longer than predefined timeout value, the connection is marked as timeout and ready to be processed later. In general, the output of this step is application layer messages.

B. Outlier Detector Training

Autoencoder is one of the deep learning architectures, it is simply another neural network. However, the difference between an autoencoder and a neural network is the target of the training. The target output of the autoencoder is not classes but the input itself and the reason for doing this is to learn the representation of the data or to remember the pattern of legitimate network traffic. To represent those traffic, we utilise byte frequencies. They are the number of occurrence of byte 0-255 in the traffic. However, the values will be greatly spread, thus we normalise these frequencies by dividing them by the message length. Let p denote the application layer message or payload of a connection. b_i is the number of occurrence of byte i in the payload p. Then x_i as a frequency of a particular byte i can be written as in equation 1. These xs then form a set of byte frequencies X of p.

$$x_i = \frac{b_i}{\text{length}(p)} \tag{1}$$

Afterward, the bytes frequencies are fed into the Autoencoder (see Figure 1). The Autoencoder consists of 256 input and output neurons because there are 256 byte frequencies.

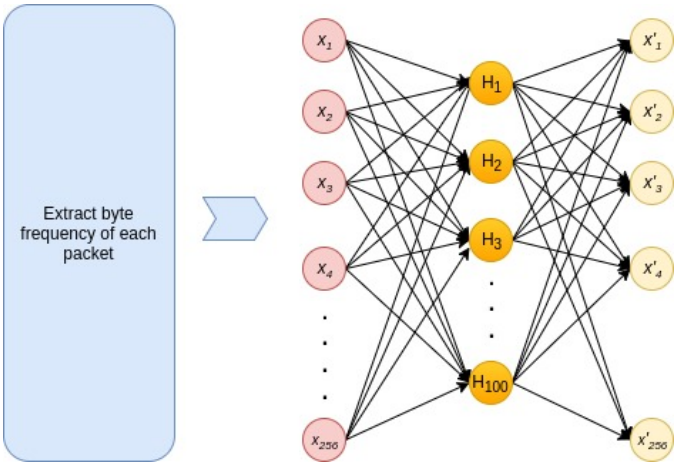

Fig. 1. Architecture of the proposed method

After the Autoencoder training is complete, we need to obtain a collection of reconstruction errors by feeding the byte frequencies of the training set to the trained autoencoder. A *reconstruction error* is an average distance between inputs and outputs, thus can be calculated using Mean Squared Error. Let e be a reconstruction error of each set of byte frequencies, then it is calculated as in equation 2.

$$e = \frac{1}{256}\sum_{i=0}^{255}(x_i - x'_i)^2 \tag{2}$$

For example, if we have ten payloads in the training set, it will have ten Xs. These Xs are used to train the Autoencoder. After the training finishes, the distance between each x and its corresponding output x' is calculated to get a set of reconstruction errors E which contains $e_1, e_2, ..., e_{10}$. This set of reconstruction errors (E) is then used to define the threshold for detecting outliers/anomalies. The process of defining thresholds will be explained in detail in section V, as it depends on the training data.

To give an illustration of how the proposed method works, Fig. 2 provides samples of good and bad input and output. The left of the image is a sample of a legitimate HTTP request along with its byte frequencies before going to the Autoencoder (top) and its respective output (bottom). The y-axis is set between 0 and 0.1 to make the plot clearer. The reconstruction error at the bottom shows that the distance between the first and second bar plot on the left is small. Whereas the right of the image shows an exploit that also contains shellcode. The output (bottom right) shows a very different distribution to the input (top right). This provides a different reconstruction error for this sample. We would expect an anomalous message to exhibit a large reconstruction error E and this forms the basis of our method.

C. Detection Phase

In the detection phase, the same preprocessing step is applied. All packets are buffered to form a full application layer message. Then we obtain byte frequencies for each application layer message to be processed by the Autoencoder. After the outputs have been obtained for the Autoencoder, they are compared to the inputs to calculate their reconstruction error, as shown in Equation 2.

TABLE II
AUTOENCODER'S HYPERPARAMETERS CONFIGURATION

Hyperparameters	Value
Number of Hidden Layer(s)	1;3;5
Number of Neurons in Hidden Layer(s)	(200); (200, 100, 200); (200, 100, 50, 100, 200)
Activation Functions in Hidden Layer(s)	ReLU
Activation Functions in Output Layer	Sigmoid
Dropout	0.2
Optimiser	adadelta
Loss Function	Binary Crossentropy
Number of Epochs	10

V. Experimental Setup and Results

We implemented the proposed system using Python 2.7.12 with Keras 2.0.6 [1], Tensorflow 1.2.1 with GPU support, and pcapy 0.10.8. All the experiments were done on a personal computer with Intel Core i7-6700 @3.40 GHz, 16 GB RAM, and NVIDIA GeForce GT-730. The autoencoder's hyperparameters configuration is shown in Table II and there are three different hidden layers configuration. The activation function in the output layer is sigmoid since we need the output to be continuous values between zero to one and we use ReLU in the hidden layer(s) because it gives faster and effective training on large datasets than Sigmoid. As an optimiser, we choose Adadelta as it has less computational overhead than the vanilla stochastic gradient descent[23]. Then, we set the number of epochs to 10 as after observing the training, the loss value starts being steady at 10 epochs. To prevent overfitting, we also add Dropout in the autoencoder, setting it to 0.2. Finally, to provide reproducibility of the result, the complete source code of our implementation can be accessed on https://github.com/bazz-066/aeids-py.

The autoencoder was trained using the traffic from 22 January 2015 of the UNSW-NB15 dataset. These traffic were fed into the autoencoder to obtain the reconstruction errors and get a threshold. This training set was also used for the implementation of previous methods including PAYL (reimplemented) and OCPAD. Hence, we can compare the results. OCPAD required different data for training and defining threshold, otherwise, it throws an exception. Thus, we split the training set with a ratio 80%:20% just for OCPAD. We considered McPAD to be part of the comparison, but its packet capturing library was unable to read PCAP files captured with Linux cooked-mode capture(SLL), while the dataset had used it to record the traffic.

To assess the performance of our proposed system and previous works, we use a combination of detection rate and false positive rate. Detection rate (DR) and false positive rate (FPR) is calculated as in equation 3 and 4. True positive (TP) is the amount of malicious traffic/packets that were detected. False positive (FP) is the amount of legitimate traffic/packets

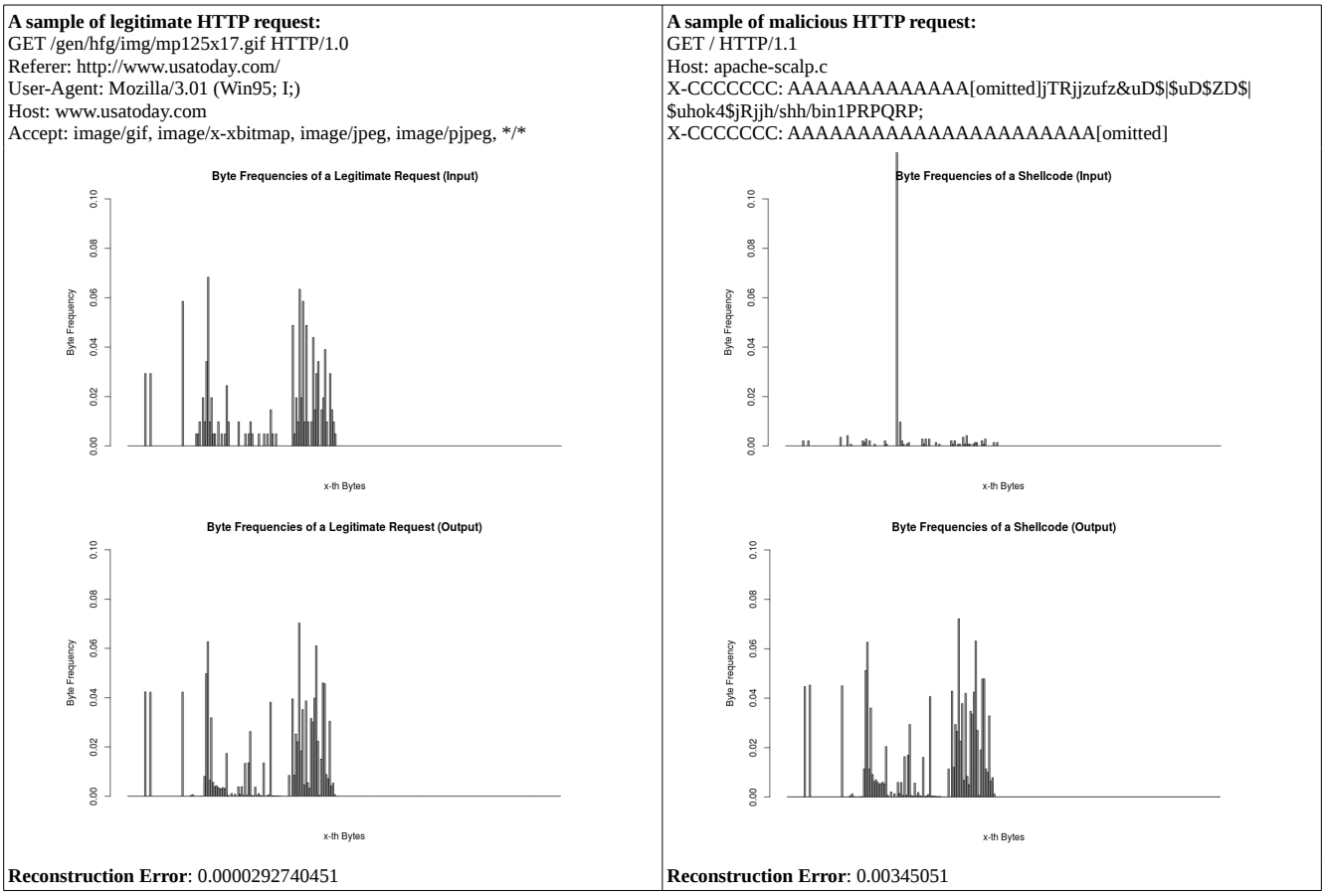

Fig. 2. Samples input and output of the autoencoder. The left box shows what the autoencoder would output when it receives a legitimate traffic. The right box shows what the autoencoder would output when malicious traffic is inserted, the output is similar to the legitimate traffic but different to the input, thus it has bigger reconstruction error.

that are considered malicious. True negative (TN) is the amount of benign traffic/packets that are considered legitimate. False negative (FN) is the amount of malicious traffic/packets that go undetected.

$$DR = \frac{TP}{TP + FN} \tag{3}$$

$$FPR = \frac{FP}{TN + FP} \tag{4}$$

In our experiments, the training set was taken from the 22 January 2015 PCAP files, while for our testing phase, we used the legitimate traffic in the 17 February 2015 and all malicious traffic from both days, apart from the Denial of Service as we are more focused on low-rate attacks. In addition to that, we only tested all methods on HTTP and SMTP, since HTTP is the most popular protocol on the internet and SMTP is the second biggest protocol in the UNSW-NB15 dataset.

A. Defining Threshold

Choosing a correct threshold for detecting outliers is an important step. The classic statistical method to achieve this is by utilising mean (μ) and standard deviation (σ). This approach works by having an assumption that outliers usually have value v, where $\mu - 2 * \sigma > v > \mu + 2 * \sigma$ or $\mu - 3 * \sigma > v > \mu + 3 * \sigma$. However, when using this approach the data must be normally distributed. Therefore, to check whether our data is normally distributed, we performed a series of statistical tests.

Firstly, the proposed system is trained using the training set. Then the same set is fed into the autoencoder to produce the reconstruction errors. It is using these reconstruction errors that we propose to use to detect outliers, so it is these that would need to be normally distributed. A histogram and boxplot of the reconstruction errors are shown in Figure 3 and 4. From those two plots, it can be seen that our data are highly skewed.

To ensure that the reconstruction errors of the training set is not normally distributed, we also run Kolmogorov-Smirnov test over those data and we got zero p-value, which means we can reject the null hypothesis that the data are normally distributed. Thus, utilising mean and standard deviation to find the threshold was not the best option for outlier detection.

Diez et al. suggest using interquartile range (IQR) to detect

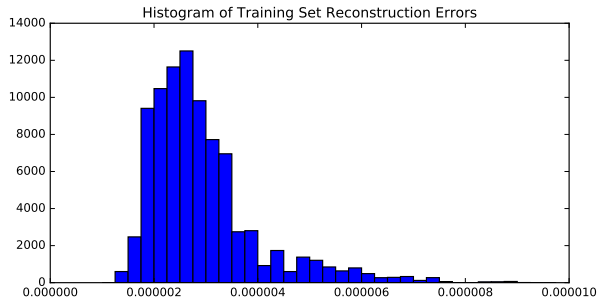

Fig. 3. Histogram of Reconstruction Errors (Section IV-B) from 22 January 2015 of UNSW-NB15 HTTP messages. It shows that the reconstruction errors of the training set are skewed.

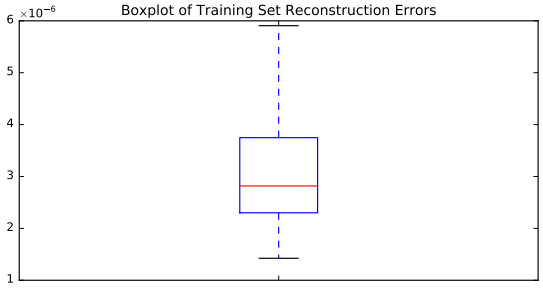

Fig. 4. Boxplot of Reconstruction Errors (Section IV-B) from 22 January 2015 of UNSW-NB15 HTTP messages that also shows how skewed the errors are.

outliers instead of using mean and standard deviation [6]. It is more robust as it relies on the median (m) which is less affected by outliers in the data. However, this approach is not suitable for skewed data. Hence, we used a modified IQR approach for skewed data proposed by Hubert et al[7]. It uses *medcouple* (MC) to measure the skewness of data. Let F be the data with a particular distribution, $MC(F)$ is medcouple of sorted F where $x_1 < x_2 < x_3 < ... < x_n$. $MC(F)$ as defined as in equation 5. x_i and x_j are sampled independently.

$$MC(F) = \underset{x_i<m<x_j}{\text{median}}\, h(x_i, x_j) \tag{5}$$

$$h(x_i, x_j) = \frac{(x_j - m) - (m - x_i)}{x_j - x_i} \tag{6}$$

Afterward, let Q_3 be the 3^{rd} quartile of the data, then T_{IQR}, the threshold of the modified IQR method is defined as in equation 7 and 8.

$$T_{IQR} = Q_3 + e^{3*MC} * 1.5 * IQR, \text{ if } MC \geq 0 \tag{7}$$

$$T_{IQR} = Q_3 + e^{4*MC} * 1.5 * IQR, \text{ if } MC < 0 \tag{8}$$

Another method of detecting outlier is by using modified Z-score which works based on median absolute deviation [8]. They argue that median and median absolute deviation are robust measures of central tendency and dispersion, respectively. However, the way it applies in this paper is different to the previous approach. The reconstruction error is not compared directly to the threshold, the error is transformed to Z-score instead. This can be achieved by calculating median absolute deviation.

Let E be a set of reconstruction errors e. Median absolute deviation MAD can be calculated as in equation 9. Thus the Z-score z of particular E_i is calculated as in equation 10[8]. When using this approach, a message is considered as malicious when its $z < 3.5$[8].

$$MAD = \text{median}(\{(|e_i - \text{median}(E)|), e_i \in E, 0 < i < n\}) \tag{9}$$

$$z = \frac{0.6745 * (|e - \text{median}(R)|)}{MAD} \tag{10}$$

TABLE III
COMPARISON OF THE AUTOENCODER IDS TO PREVIOUS WORKS

Method	DR (%)	FPR (%)
Autoencoder (T_{IQR} Threshold)	68.91	0.99
Autoencoder (Z-Score Threshold)	100	8.01
OCPAD (1-gram)	19.88	0.00
OCPAD (3-gram) (HTTP only)	29.08	8.85
PAYL	36.1	0.05

TABLE IV
DETECTION RATE (%) OF PROPOSED WORK, PAYL, AND OCPAD FOR EACH ATTACK CATEGORY.

	Auto encoder T_{IQR}	Auto encoder Z-Score	PAYL	OCPAD (1-Gram)	OCPAD (3-Gram)
Analysis	1.60	100.00	0.43	0.00	25.47
Back-doors	11.11	100.00	8.70	1.69	38.98
Exploits	47.51	100.00	87.12	10.53	35.63
Fuzzers	97.49	100.00	4.16	0.04	9.85
Generic	33.86	100.00	49.26	2.49	19.16
Recon-naissance	6.11	100.00	0.23	0.23	27.17
Worms	81.12	100.00	26.49	4.11	24.76

B. Results and Discussion

To test the Autoencoder we used two datasets. To get the false negative results we used malicious traffic from both days of the UNSW-NB15 dataset, as discussed in Section III. It has a collection of 17041 malicious HTTP connections and 4631 malicious SMTP connections. To get the false positive rate we used legitimate traffic from 17 February 2015 of the UNSW-NB15 dataset. There are 131842 HTTP and SMTP connection in this set.

This experiment also tested our system with two different threshold values - T_{IQR}, and Z-score-based threshold. All methods, including the previous works, are trained on the same training set (22 January 2015) and tested using the same testing sets containing malicious traffic. Table III shows the result of our experiments.

Our proposed system using the Autoencoder achieved 100% detection rate at best with 8.01% false positive rate. This performance was achieved by using Z-Score as the threshold. To understand better why our proposed method can achieve this performance, we also plotted the reconstruction errors of

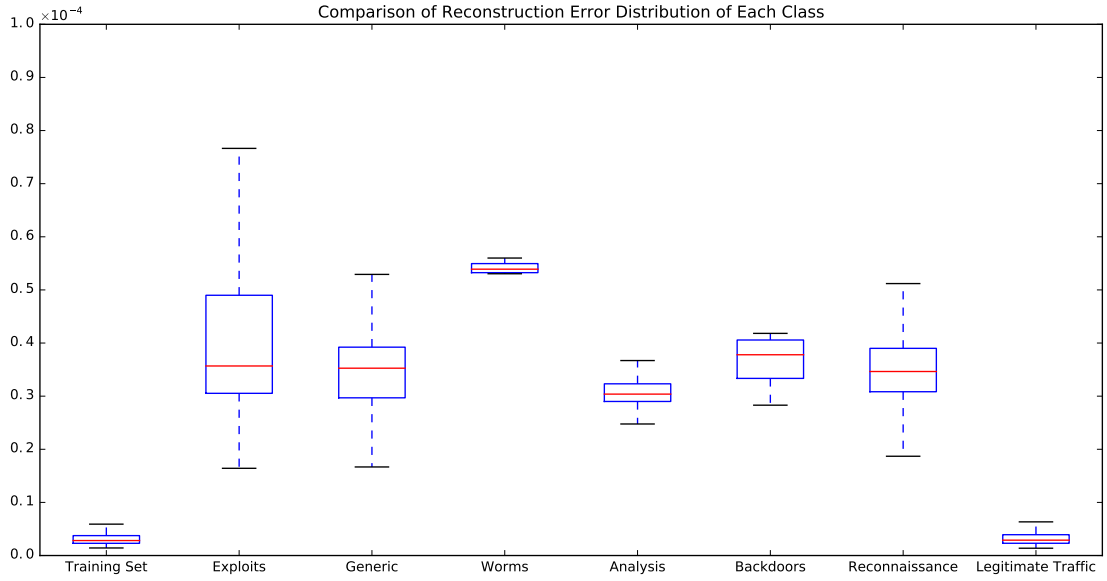

Fig. 5. Boxplot of Reconstruction Errors (Section IV-B) from 22 January 2015 of UNSW-NB15 HTTP messages. It shows the distribution of the Reconstruction Errors of legitimate and malicious traffic where the malicious one tends to have higher errors.

all tested classes on Figure 5, except the Fuzzers since their errors are massive and very distant from the others and would make the plot unclear. It can be seen that, in general, the reconstruction errors of both the training set and the legitimate traffic in the testing set are lower than the malicious traffic reconstruction errors.

When we analysed the false positives manually, we found that the false positives of HTTP traffic contained arbitrary characters in their Host header. In the SMTP traffic, the false positives are likely due to the presence of strings similar to Unix commands or assembly code.

We also analysed the result of using T_{IQR} as a threshold for detecting HTTP-based attacks as the detection rate varies among different kind of attacks. Worms and Fuzzers are the most easily detectable attacks since those two have the highest reconstruction errors as shown in Figure 5. Fuzzers have many repeated characters and Worms contain Unix commands to instruct the server to perform an action. Exploits were second best. The longer exploits with actual code were most likely to be detected, while some exploit attacks in the dataset were missed due to having short requests and no code. Analysis, Generic, and Reconnaissance have similar behaviour. They were missed when using T_{IQR} as the threshold because their HTTP messages simply contain requests to files or web pages. They are similar to legitimate requests and have lower reconstruction errors than exploits, worms, and fuzzers. However, these undetected attacks were detected when we changed to Z-score-based threshold method.

To get more detail in how the performance of our proposed method compared to the previous works, we also analysed the detection rates of all methods for each attack category as shown in Table IV. The T_{IQR} method works best for detecting Fuzzers and Worms, while the Z-Score method is able to detect all attack categories. PAYL gives a better result than T_{IQR} on detecting exploits. OCPAD gives no more than 50% detection rate for all kind of attacks, which makes it perform the worst among others.

From the experiment, it can also be seen that our proposed method surpassed all previous works. OCPAD was not able to run over SMTP when we tried 3-gram. It was using too much RAM, even with 16 GB memory for the Java Virtual Machine, we still got the OutOfMemory exception. PAYL works as intended, but the reason its detection rate dropped could be due to not reconstructing TCP segments. SMTP messages are big and normally distributed into few network packets.

We also ran an experiment to see the effect of different hidden layers configurations and the result shows when the number of hidden layers is more than three, the autoencoder tends to be overfitting as the false positive rate increases. This behaviour occurred when either T_{IQR} or Z-Score were being used as shown in Table VI. Thus we can conclude that 200-100-200 configuration is the most suited for this case.

The last parameter that may affect the performance of our proposed method is the timeout. Our TCP reassembly has a predefined timeout value to determine when a TCP stream has been in the buffer for too long. Therefore we tried giving three timeout values with two hidden layers. As can be seen in Table V, the small timeout does not give any improvement in the detection rate. It worsened the false positive rate instead. This behaviour might occur due to early termination of TCP streams while they should have waited longer for other packets to come, thus making the proposed system did not see the

TABLE V
RESULTS OF VARIOUS TIMEOUT VALUES TO DETECTION AND FALSE POSITIVE RATES

	T_{IQR}		Z-Score	
Timeout (s)	**DR (%)**	**FPR (%)**	**DR (%)**	**FPR (%)**
2	68.73	1.355	100	9.04
5	68.265	0.445	100	8.435
10	68.205	0.315	100	8.005

TABLE VI
THE EFFECT OF VARIOUS HIDDEN LAYERS CONFIGURATIONS

Number of Neurons in Each Hidden Layer	**Threshold Method**	**DR (%)**	**FPR (%)**
200	T_{IQR}	68.91	0.99
200-100-200	T_{IQR}	68.21	0.63
200-100-50-100-200	T_{IQR}	69.21	3.16
200	Z-Score	100	8.11
200-100-200	Z-Score	100	8.01
200-100-50-100-200	Z-Score	100	8.1

whole information.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we have presented a method to detect low-rate attacks such as exploits, backdoors, fuzzers, and worms. Autoencoder, combined with the statistical thresholding approach, was able to perform outlier detection. On the contrary, all previous works that had been tested could not match the proposed system performance. The proposed system surpassed all previous works, in term of detection rate with a compensation of increasing false positive rate.

Based on the experiment, modified IQR method to detect outliers is the most cautious at selecting outlier, resulting in the lower false positives than the Z-score method, while still keeping the detection rate above 65%. However, the Z-score method gave better detection rate with increasing false positive rate and research toward decreasing false positive while still utilising autoencoder as outlier detector might also be a future work. This also includes the possibility of leveraging other text mining related features such as n-gram and analysing the application layer protocol deeper, thus each protocol may have a specific set of features.

ACKNOWLEDGMENT

This work was supported by the Indonesia Endowment Fund for Education (Lembaga Pengelola Dana Pendidikan/LPDP) under the grant number PRJ-968/LPDP.3/2016 of the LPDP.

REFERENCES

[1] "Keras: The python deep learning library." [Online]. Available: https://keras.io/

[2] "Transmission control protocol." [Online]. Available: https://tools.ietf.org/html/rfc793

[3] D. Ariu, R. Tronci, and G. Giacinto, "Hmmpayl: An intrusion detection system based on hidden markov models," *computers & security*, vol. 30, no. 4, pp. 221–241, 2011.

[4] D. Bolzoni, S. Etalle, and P. Hartel, "Poseidon: a 2-tier anomaly-based network intrusion detection system," in *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*. IEEE, 2006, pp. 10–pp.

[5] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6, pp. 353–375, 2011.

[6] D. M. Diez, C. D. Barr, and M. Cetinkaya-Rundel, *OpenIntro statistics*. CreateSpace, 2012.

[7] M. Hubert and E. Vandervieren, "An adjusted boxplot for skewed distributions," *Computational statistics & data analysis*, vol. 52, no. 12, pp. 5186–5201, 2008.

[8] B. Iglewicz and D. C. Hoaglin, *How to detect and handle outliers*. Asq Press, 1993, vol. 16.

[9] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "Repids: A multi tier real-time payload-based intrusion detection system," *Computer Networks*, vol. 57, no. 3, pp. 811–824, 2013.

[10] A. Jamdagni, Z. Tan, P. Nanda, X. He, and R. Liu, "Intrusion detection using geometrical structure," in *2009 Fourth International Conference on Frontier of Computer Science and Technology*. IEEE, 2009, pp. 327–333.

[11] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26.

[12] M. Kakavand, N. Mustapha, A. Mustapha, and M. T. Abdullah, "Effective dimensionality reduction of payload-based anomaly detection in tmad model for http payload." *KSII Transactions on Internet & Information Systems*, vol. 10, no. 8, 2016.

[13] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 376–385.

[14] Mitrecnd, "Pynids," Jul 2014. [Online]. Available: https://github.com/MITRECND/pynids

[15] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE, 2015, pp. 1–6.

[16] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "Mcpad: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[17] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems," in *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 2006, pp. 488–498.

[18] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.

[19] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[20] M. Swarnkar and N. Hubballi, "Ocpad: One class naive bayes classifier for payload based anomaly detection," *Expert Systems with Applications*, vol. 64, pp. 330–339, 2016.

[21] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 226–248.

[22] K. Wang and S. J. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," in *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15 - 17, 2004. Proceedings*, E. Jonsson, A. Valdes, and M. Almgren, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 203–222. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30143-1{_}11

[23] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.