


# Application of Representation Learning-Based Chronological Modeling for Network Intrusion Detection

Nitin O. Mathur, University of Cincinnati, USA\*

 <https://orcid.org/0000-0001-8590-2879>

Chengcheng Li, University of Cincinnati, USA

Bilal Gonen, University of Cincinnati, USA

Kijung Lee, University of Cincinnati, USA

## ABSTRACT

An autoencoder has the potential to overcome the limitations of current intrusion detection methods by recognizing benign user activity rather than differentiating between benign and malicious activity. However, the line separating them is quite blurry with a significant overlap. The first part of this study aims to investigate the rationale behind this overlap. The results suggest that although a subset of traffic cannot be separated without labels, timestamps have the potential to be leveraged for identification of activity that does not conform to the normal or expected behavior of the network. The second part aims to eliminate dependence on visual-inspections by exploring automation. The trend of errors for HTTP traffic was modeled chronologically using resampled data and moving averages. This model successfully identified attacks that had orchestrated over HTTP within their respective time slots. These results support the hypothesis that it is technically feasible to build an anomaly-based intrusion detection system where each individual observation need not be categorized.

## KEYWORDS

Autoencoder, Brute-Force, CICIDS2017, Data Resampling, Denial-of-Service, Infiltration, Loss Function, Moving Average, Neural Networks, Root Mean Squared Error, Timestamps, Zero-Day-Vulnerability

## INTRODUCTION

Monitoring any system or process is done by collecting and examining the relevant metrics. An Intrusion Detection System (IDS) is used to monitor a computing infrastructure and identify potential malicious activity. Malicious activity can be defined as anything that is a threat to the confidentiality, integrity, or availability of data. It usually occurs when an uncorrected vulnerability is exploited. It may range from unauthorized access to viruses and denial-of-service attacks. Modern computing infrastructure can generate several gigabytes of metrics within a short span of time, and real time analysis of these metrics is essential for timely identification of such threats.

Intrusion detection methods can be broadly classified as signature-based or anomaly-based. The former involves maintaining a database of known threats and their behavior. Intrusions can be recognized by comparing current activity to these known patterns. This method is often used in commercial products due to its high accuracy (Bace & Mell, 2001). These systems are very efficient

DOI: 10.4018/IJISP.291701

\*Corresponding Author

This article published as an Open Access Article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

at identifying known threats, but they cannot detect new and emerging threats that are orchestrated by exploiting zero-day-vulnerabilities (Panda & Patra, 2009). The anomaly-based method is data driven. An anomaly is a data point that does not conform to the normal or expected behavior of a system (Chandola, et al., 2009). Anomaly-based intrusion detection using supervised classifiers have shown excellent results in the literature, particularly with Random Forest (RF) based models (Choudhury & Bhowal, 2015), (Anbar, et al., 2016), (Chang, et al., 2017), (Obeidat, et al., 2019), (Patil, et al., 2019). However, the process of acquiring accurate labels, keeping them updated, and accounting for any imbalance in the training data can be expensive and time consuming (Chandola, et al., 2009). Unsupervised learning can address most of these issues, but with higher false alarm rates (Davis & Clark, 2011). Semi-supervised methods and their advantages have also been widely discussed in the literature (Dong & Wang, 2016), (Aung & Min, 2017), (Heba, et al., 2010), but they do not overcome the inherent limitations of supervised classification.

To address the aforementioned issues, related work in the literature (Choi, et al., 2019), (Catillo, et al., 2019) has proposed the application of a specific kind of feed-forward, non-recurrent artificial neural network known as the *autoencoder*. The basic methodology involves modelling benign user activity and evaluating future activity with respect to its deviation from this learned model by measuring reconstruction errors of the autoencoder. The reconstruction errors of benign and malicious activities usually follow normal distributions. However, they overlap with each other making it challenging to determine an appropriate classification threshold.

## CONTRIBUTION

This work presents a new perspective to the autoencoder-based model by leveraging *timestamps*, which have often been ignored in the literature. The proposed solution does not eliminate the overlap or categorize each individual packet or flow. Instead, it identifies deviations from the expected behavior of the network by tracking consistent irregularities in autoencoder reconstruction errors over a period of time.

## CICIDS2017 DATASET

The CICIDS2017 Dataset (Sharafaldin, et al., 2018) was used for this study. It is an IDS evaluation dataset generated by the Canadian Institute of Cybersecurity based at the University of New Brunswick in Fredericton, Canada. This dataset was generated over a period of five days from *Monday, July 3, 2017* through *Friday, July 7, 2017*. It has been available for use by the research community since 2018 in Packet Capture (PCAP) format, as well as in CSV format where each record is a labelled flow with 84 features. Monday is the first day and includes benign traffic only. Simulated attacks were executed from Tuesday through Friday. Each record in the CSV files is a traffic flow (Claffy, et al., 1989), (Quittek, et al., 2004) with some measured properties and calculated statistics of that flow. The CICFlowMeter tool (Habibi Lashkari, 2018), (Lashkari, et al., 2017), (Draper-Gil, et al., 2016) was used to generate these flows and the calculated features from the PCAP files. Table 1 summarizes the CSV files. This dataset is a good match for this study for the following reasons:

1. *Monday's* data contains only *benign* traffic, which was used to train the models.
2. This dataset includes *timestamps*, which are an important aspect of this work.

Table 1. Dataset Summary

CSV	Benign Flows	Malicious Flows	Network Activity
Monday	529918	0	Benign (Normal user activities)
Tuesday	432074	13835	Benign + Brute force (FTP-Patator, SSH-Patator)
Wednesday	440031	252672	Benign + DoS (Slowloris, Slowhttptest, Hulk, GoldenEye), Heartbleed
Thursday-Morning	168186	2180	Benign + Web attacks (Brute force, XSS, SQL-Injection)
Thursday-Afternoon	288566	36	Benign + Infiltration
Friday-Morning	189067	1966	Benign + Botnet (ARES)
Friday-Afternoon-Portscan	127537	158930	Benign + Portscan
Friday-Afternoon-DDoS	97718	128027	Benign + DDoS

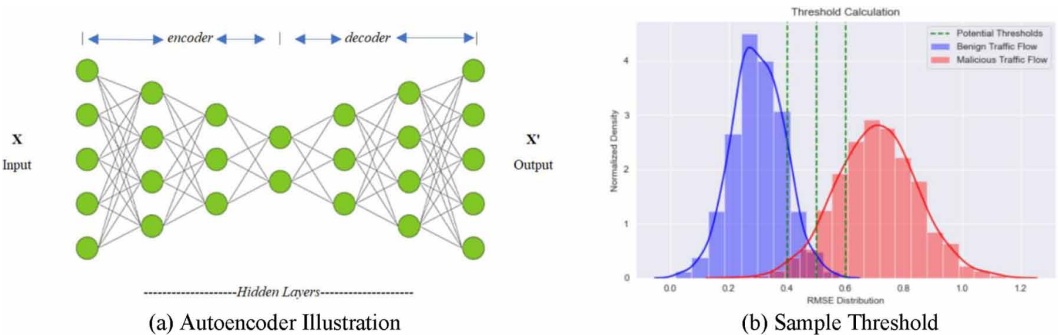
BACKGROUND AND RELATED WORK

Autoencoders were introduced in the 1980s based on the principle of learning internal representations by error propagation (Rumelhart, et al., 1985). An autoencoder is a self-learning artificial neural network where the input and output layers have the same number of nodes, whereas the middle layer known metaphorically as the bottleneck has fewer nodes. Conceptually, an autoencoder resembles an hourglass. Functionally, it consists of an encoder and a decoder as illustrated in sub-figure (a) of Figure 1.

The encoder reduces unlabeled input data (X) from its superficial to intrinsic dimensions. In other words, it transforms the data from higher dimensional space to a lower dimensional representation (h). The decoder then reconstructs this compressed data back to the original number of dimensions. The reconstructed data (X') is similar to the original (X), but it is not an exact match. The difference between (X) and (X') is the reconstruction error. This error is measured using a loss function, such as mean squared error. During the training process, the reconstruction error is back-propagated through the network after each epoch, and the autoencoder updates the weights and biases to minimize the reconstruction error. By doing so, the autoencoder models the identity function of the input data distribution to arbitrary accuracy. This concept of identity mapping using an autoencoder was demonstrated in the early 1990s by (Kramer, 1991).

An autoencoder can be used for inference after it is trained on historical data that is confirmed to be normal. The trained model is capable of reconstructing new, never-seen-before instances of data

Figure 1. Autoencoder as an anomaly detector



that belong to the same distribution. When future data is passed through it, the reconstruction error is measured. If the error is greater than a predefined threshold, it implies that the autoencoder has not seen that activity before and it is likely to be a deviation from the normal or expected behavior of the system. (Japkowicz, et al., 1995) applied this methodology to three diverse applications: CH46 helicopters gearbox fault detection, identification of a specific sequence of DNA known as the promoter, and sonar target recognition. (Thompson, et al., 2002) investigated anomaly detection using an autoencoder with data from the values of average CPU utilization of a designated server over a period of 10 weeks. More recently, (Sakurada & Yairi, 2014) used spacecraft telemetry data consisting of continuous sensor measurements to demonstrate the application of an autoencoder as the core component for anomaly detection based on comparison of reconstruction errors. Although the studies referenced above are not related to network intrusion detection, the methodology is generic and can be applied to any domain. Within the domain of cybersecurity, autoencoders have primarily been used for feature engineering to improve the performance of supervised classifiers. In one such study by (Javaid, et al., 2016) on the NSL-KDD (Tavallae, et al., 2009) dataset, an autoencoder was used to generate a new feature representation. These new features along with the original label vectors were used for the final classification using Softmax Regression. (Yousefi-Azar, et al., 2017) analyzed the performance of various classifiers with the original feature set, and then with a set of latent features generated using an autoencoder. (Shone, et al., 2018) proposed stacked non-symmetric deep autoencoders combined with Random Forest for the final classification. Daisy chaining of four shallow autoencoders was implemented by (Farahnakian & Heikkonen, 2018). The shallow autoencoders were trained sequentially. The last layer was a supervised layer that used a SoftMax classifier for the final output. (Mirsky, 2018) proposed a plug and play Network Intrusion Detection System called Kitsune which used an ensemble of autoencoders to collectively differentiate between normal and abnormal network traffic patterns with a performance comparable to offline anomaly detectors. The system used incremental statistics maintained over a damped window. (Zavrak, 2020) evaluated a Deep Autoencoder and a Variational Autoencoder with One Class Support Vector Machine (OCSVM) using the CICIDS2017 (Sharafaldin, et al., 2018) dataset. Both autoencoders had five hidden layers with the architecture: (79-512-256-64-256-512-79). The classification results were analyzed using Receiver Operating Characteristics (ROC) and Area under ROC curve (AUC). The results were compared with OCSVM. Overall, the Variational Autoencoder showed better performance, but with a high false-positive rate. (Choi, et al., 2019) developed a network intrusion detection system with an autoencoder as the core component. This model was evaluated on the NSL-KDD (Tavallae, et al., 2009) dataset. A small percentage of abnormal data (records labelled as attacks) were included in the training data to reflect real world conditions, where some abnormal patterns exist within normal data. The classification threshold was defined as a function of the percentage of this abnormal data. (Catillo, et al., 2019) explored the use of a deep autoencoder to identify denial-of-service attacks using Wednesday's subset of the CICIDS2017 (Sharafaldin, et al., 2018) dataset. The input and output layers of the autoencoder consisted of 83 neurons because the features *Flow\_ID* and *Timestamp* were dropped. There were five hidden layers, and the structure of the autoencoder was (83-100-90-10-90-100-83). Wednesday's dataset was split into training (40%, benign data only), validation (20%, benign data only), threshold (20%, both benign and malicious) and testing (20%, both benign and malicious) subsets. The threshold subset was used to determine the probability that maximized the F1 score, which was used for the final classification of the testing data. The same group extended this study in (Catillo, et al., 2020) to reduce the number of false alerts with an updated version of the same dataset (CSE-CIC-IDS2018) using a double learning approach. In addition to the primary autoencoder, two secondary autoencoders were deployed. The first autoencoder at the second level was used to separate benign flows from False Negatives (FN), and the second one was used to separate attacks from False Positives (FP).

## The Problem Statement

The application of representation learning with an autoencoder at the heart of it successfully overcomes the limitations of signature-based and anomaly-based intrusion detection methods. But it has one major drawback. The trained autoencoder encodes a subset of normal and abnormal data equally well, which causes the reconstruction error distributions of both categories to overlap as demonstrated in histogram (b) of Figure 1. Selecting a specific probability as the classification threshold ( $\theta$ ) depends on whether the organization's stakeholders prioritize precision or recall. The threshold is a critical component because the performance of the model is dependent on it (Choi, et al., 2019). (Japkowicz, et al., 1995) proposed a semi-automated threshold determination component. For cases where the separation of normal and abnormal data was unclear, this component constructed absolute and intermediate positive and negative regions within the epoch versus reconstruction error space. (Catillo, et al., 2019) selected a threshold that maximized the F1 score, which is the harmonic mean of precision and recall. (Choi, et al., 2019) added some abnormal data to the training set and defined the threshold as a function of the anomalous data percentage. However, irrespective of the threshold selection method, this overlap results in an unreasonable number of false alerts and missed anomalies.

In line with the problem statement, the research questions addressed in this study are the following:

1. Why does an autoencoder that is trained on data generated by benign network traffic encode a subset of both benign and malicious traffic equally well, thus causing an overlap in their reconstruction error distributions?
2. What effect does an increase in the complexity of the neural network have on this overlap?
3. What effect does a decrease in the diversity within the input data have on this overlap?

The following hypotheses were inspired by '*The Adventures of Sherlock Holmes*' written by Sir Arthur Conan Doyle (Doyle, 1859-1930). As an analogy to the postal system in the physical world, when a package is dispatched for delivery, only the information on the envelope is accessible to an observer (for example, the addresses and dispatch date). Although, the contents of the package are not accessible when it is in transit, the observer can deduce the purpose of the exchange from the frequency, direction and size of the packages using the methods of Sherlock Holmes. However, for some packages, a visual inspection of the envelope is not sufficient, and the contents or purpose of the exchange may remain ambiguous. Comparably, in the virtual world, when malicious activities within the application layer are being examined at the network and transport layers, a subset of traffic cannot be categorized without explicit labels.

1. A subset of network traffic originating due to malicious activity in the application layer is similar in structure to benign traffic when observed in the lower layers and cannot be differentiated without explicit labels regardless of the complexity of the neural network or the diversity within the data.
2. Instead of the traditional method of categorizing each individual packet or flow, a deviation from the expected behavior of the network can be identified by statistically modelling reconstruction errors of the trained autoencoder in chronological order.

## Drawbacks of Classifying Each Individual Observation

A traffic flow (Claffy, et al., 1989), (Quittek, et al., 2004) consists of statistics summarized from several packets. However, a flow is not necessarily mapped 1:1 to a specific transport connection (Rajahalme, et al., 2004). For instance, the three records shown in Table 2 belong to Tuesday's subset of the CICIDS2017 (Sharafaldin, et al., 2018) dataset. Although, they belong to the same TCP connection, they exist as three separate records with the first two labelled as malicious (FTP-Patator brute-force attack) and the third one labelled as benign. Similarly, a single packet may not

be malicious in isolation, but it could be part of a structured denial-of-service attack. Therefore, we suggest that building a model to classify each individual observation is not an ideal solution for network intrusion detection.

Table 2. Traffic Flow

Source IP	Source Port	Destination IP	Destination Port	Protocol	Time stamp	Flow Duration	Total Fwd Packets	Label
172.16.0.1	53734	192.168.10.50	21	6	9:31 AM	4	2	FTP-Patator
172.16.0.1	53734	192.168.10.50	21	6	9:31 AM	9306763	9	FTP-Patator
192.168.10.50	21	172.16.0.1	53734	6	9:31 AM	31063	2	Benign

### Other Considerations: Random Splitting of Data Into Train and Test Sets Without a Resampling Procedure

Randomized splitting of data into training and testing subsets is done with the *stratify* parameter set to the target (label). However, other variables can have a significant role in the formation of distinct substructures within the data and can influence the final results. These substructures can be visualized using a two-dimensional representation of the data. As an example, Wednesday's data from the CICIDS2017 (Sharafaldin, et al., 2018) dataset is projected in scatter plot (a) of Figure 2.

Wednesday's data was reduced to 20 vectors which covered 99.66% of the variance as shown in plot (b) of Figure 2. The first two vectors of this compressed data were used to plot the data. Scikit-learn's (Pedregosa, et al., 2011) Principal Component Analysis (PCA) module was used for this

Figure 2. Data Sub-structures

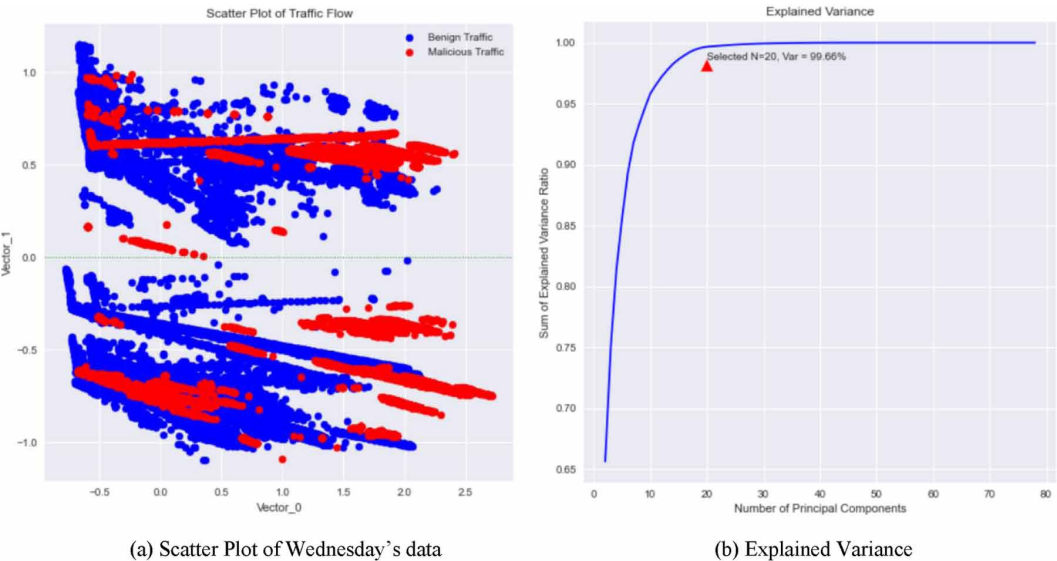


Table 3. Top 10 Feature Weights - Vector\_0

Feature Name	Weight in Vector_0	Rank
Fwd IAT Total	0.331933	1
Flow Duration	0.331787	2
Idle Max	0.315553	3
Fwd IAT Max	0.315068	4
Flow IAT Max	0.314738	5
Idle Mean	0.311927	6
Idle Min	0.308521	7
Fwd IAT Std	0.183275	8
Avg Bwd Segment Size	0.151220	9
Bwd Packet Length Mean	0.151220	9

activity. Table 3 and Table 4 show the weights of the top 10 features in the first two vectors (Principal Components). The binary feature *ACK Flag Count* was found to hold a weight of 0.78 in the second vector and had considerable influence in defining the substructures seen in scatter plot (a) of Figure 2.

In this work, we examined two use-cases using the methodology proposed by (Catillo, et al., 2019) for identifying denial-of-service attacks. In use-case 1, 95% of the *benign* and *malicious* flows had opposite values for *ACK Flag Count*, while in use-case 2, 95% had the same values. The results summarized using traditional metrics in Table 5 and Table 6 show that the distribution of use-case 1 where majority of the *benign* and *malicious* flows had opposite values for *ACK Flag Count* yielded better results. In the current study, the entire set of each day's traffic as it transpired naturally was used without splitting it into train, validate and test subsets in order to eradicate this bias.

Table 4. Top 10 Feature Weights - Vector\_1

Feature Name	Weight in Vector_1	Rank
ACK Flag Count	<b>0.780023</b>	1
Destination Port	0.192774	2
URG Flag Count	0.165120	3
Fwd Push Flags	0.102586	4
SYN Flag Count	0.102586	4
Fwd Packet/s	0.050348	6
Idle Min	0.038528	7
Idle Mean	0.036184	8
Idle Max	0.033159	9
Fwd IAT Max	0.032073	10

Table 5. Use-Case 1 Results

Metric	Formula	Value
Precision	$TP / (TP+FP)$	<b>0.9998</b>
Recall	$TP / (TP+FN)$	1.0
Accuracy	$(TP+TN) / (TP+TN +FP+FN)$	0.9999
False Alarm Rate	$FP / (FP+TN)$	6.36e-05
F1 Score	$2*(Precision * Recall)/(Precision + Recall)$	0.9999

## DETERMINING AN APPROPRIATE AUTOENCODER CONFIGURATION

The number of neurons in the input and output layers of the network are defined by external problem specifications (Hagan, et al., 2014). The CICIDS2017 (Sharafaldin, et al., 2018) data is composed of eight sets as summarized in Table 1. Each record has 84 network flow features and a label.

### Data Preparation – Phase 1

#### *Feature Selection and the Curse of Dimensionality*

Although eliminating features based on variance and internal correlation improves the time and space complexity of any learning algorithm, it is possible to miss out on an anomaly if traffic characterizations change in the future. We therefore suggest that for dynamic data, it is more appropriate to project the entire dataset as it occurs naturally to a lower dimensional representation instead of dropping features outright. However, the following features were dropped from the training data for the reasons stated below:

1. **Source and Destination IP Addresses:** Encoding IP addresses to an integer representation and including them in the training data can influence the trained model to identify malicious activity based on the source, thereby narrowing down the generalizability of the solution. The infrastructure used to generate the CICIDS2017 Dataset (Sharafaldin, et al., 2018) consisted of a separate attack-network that was used to orchestrate the attacks on the victim-network. The source IP address on all the records labelled as malicious are therefore 172.16.0.1, which is the internal IP address of the victim's firewall (An exception to this is 'Infiltration', as explained under the section- 'Infiltration and Portscan'). Moreover, IP addresses can be spoofed and/or are likely to change if they were assigned by a DHCP Server.
2. **Flow ID:** It is a unique string identifier given to each flow. It consists of a string created by concatenating the source and destination IP addresses and port numbers. Including the Flow ID would have had the same influence as including the IP addresses.

Table 6. Use-Case 2 Results

Metric	Formula	Value
Precision	$TP / (TP+FP)$	0.9449
Recall	$TP / (TP+FN)$	0.8385
Accuracy	$(TP+TN) / (TP+TN +FP+FN)$	0.9536
False Alarm Rate	$FP / (FP+TN)$	0.0137
F1 Score	$2*(Precision * Recall)/(Precision + Recall)$	0.8885



3. **Timestamps:** Encoding timestamps to numeric values and including them in the training set can skew the model to classify attacks based on the time slots in which they were orchestrated. Timestamps were excluded from the training data, but they were stored in another variable for later use.
4. **Fwd Header Length:** This feature occurs twice with the same values. One duplicate instance was dropped.

#### Records

1. Records with *Infinity* and *NaN* values were removed.
2. Duplicate records were removed for this preliminary analysis (The count of unique records is displayed in Figure 3 for reference).
3. Labels were made binary – *Benign* (0) or *Malicious* (1).

The resultant dataset had 79 features and the ‘Label’ column. Labels were not included in the training data. The input and output layers therefore had 79 neurons. An important question was, “*How to ascertain the required number of middle layers and the optimal number of neurons in each layer?*”. Determining the optimal number of neurons in the hidden layer is still an active area of research (Hagan, et al., 2014). Theoretically, a neural network with one hidden layer is capable of universal approximation (Heaton, 2015). As a first step, one middle layer with an arbitrary number of neurons (10) was chosen. The structure of the autoencoder was therefore (79-10-79):

- Input Layer - 79 Neurons
- Middle Layer - 10 Neurons
- Output Layer - 79 Neurons

#### Data Scaling

Scikit-learn’s (Pedregosa, et al., 2011) *MinMaxScaler()* function shown in Equation 1, was used to scale the features in the range of 0 and 1.

Equation 1. MinMaxScaler

$$x_{scaled} = \frac{x - \text{Min}(x)}{\text{Max}(x) - \text{Min}(x)}$$

Monday’s dataset that contains only benign traffic was used for training. It was scaled using the *fit\_transform()* method. The datasets from Tuesday to Friday were used for testing. The training parameters from Monday’s dataset were reused to scale the testing data using the *transform()* method. This was done to treat the test data as new, never-seen-before data.

#### Libraries

*Tensorflow* (Abadi, et al., 2015) version 2.3.0 and *Keras* (Chollet & others, 2015) version 2.4.0 libraries were used to build the Autoencoder.

## Other Parameters

1. **Activation Function:** A common activation function, as suggested by most current literature, and used in this work is the Rectified Linear Unit (*ReLU*) function. Over the last decade, *ReLU* has become a de-facto standard (Heaton, 2015). The *ReLU* function is defined in Equation 2. If the input ( $z$ ) is positive, the function will return ( $z$ ), else it will return zero.  
Equation 2. ReLu Activation Function

$$R(z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

2. **Loss Function:** The autoencoder minimized the Mean Squared Error (MSE) with each iteration during the training process. The *Sqrt()* function of NumPy (Harris, et al., 2020) was used to compute the Root Mean Squared Error (RMSE) for the analysis. The RMSE of matrices  $X$  and  $X'$  is defined in Equation 3, where  $n$  is the number of observations.  
Equation 3. Root Mean Squared Error

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (X' - X)^2}$$

3. **Training Epochs:** During the training process, the MSE always plateaued well before 100 epochs. Therefore, each autoencoder was trained for 100 epochs in the rest of this study.
4. **Batch Size:** As recommended by (Heaton, 2015), an initial batch size equal to 10% of the number of records was used.
5. **Optimizer Function:** The *Adam* Optimizer (Kingma, 2014) was used with default hyper parameters. This optimization algorithm is computationally more efficient since it dynamically adjusts the learning rate over the course of the training process.

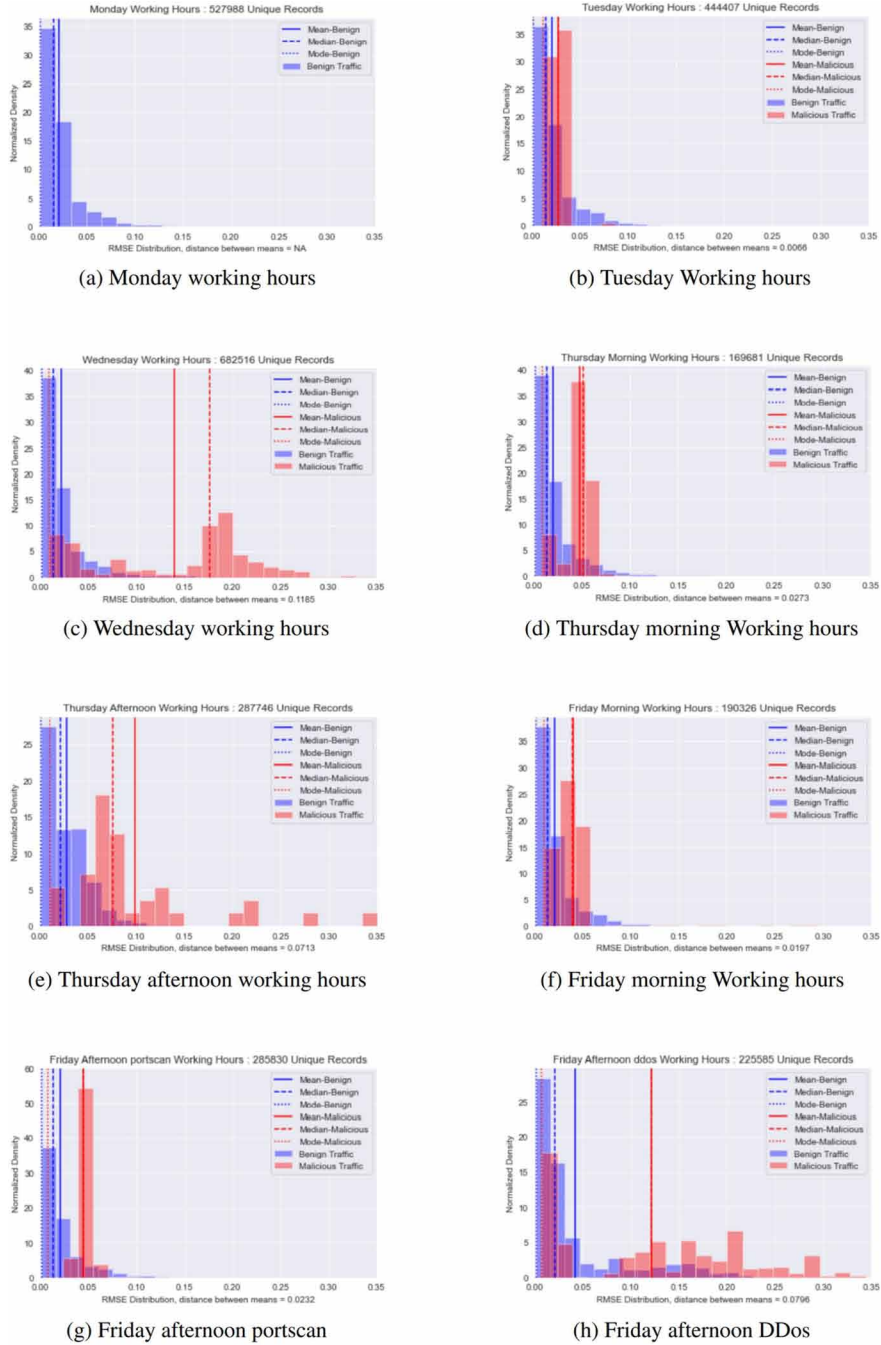
## Evaluation of the Autoencoder with one middle layer (79-10-79) on the CICIDS2017 Dataset

As explained in the section on ‘Data Scaling’, the autoencoder was trained on the entire dataset of Monday and was tested on the eight datasets described in Table 1, including Monday’s data on which it was trained. The RMSE distributions of the eight sets are plotted in Figure 3. These histograms show that benign flows follow a consistent distribution but have a significant overlap with malicious flows. Wednesday’s data, which contains denial-of-service attacks exhibits the best separation.

## The Effects of Varying the Number of Neurons in the Middle Layer

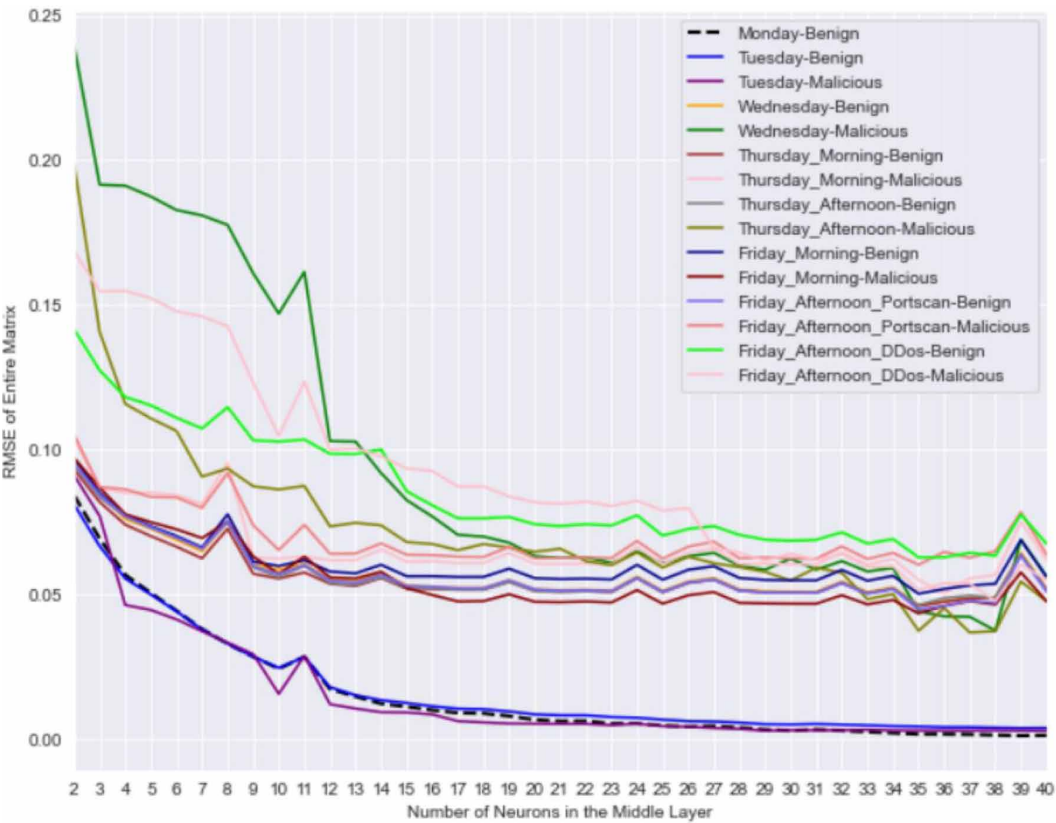
As mentioned earlier, determining the optimal number of neurons in the hidden layer is still an active area of research (Hagan, et al., 2014). In this interesting experiment, the impact of varying the number of neurons in the hidden layer was studied. The hidden layer had 2 neurons to start with. The count was increased by one neuron with each iteration till there were 40 neurons, which is around half the number of neurons in the input and output layers. The model was retrained during each iteration. The impact of the additional neurons on the subsets are plotted in Figure 4 for reference. In the interest of space, each data point in the plot represents the RMSE of the entire matrix of flows in its respective subset. In the printed version of this paper, this image may appear in grayscale and the colors may not be discernable. But the colors are not relevant. The important observations are the following:

Figure 3. RMSE Distribution of the CICIDS2017 Dataset (Autoencoder Structure: 79-10-79)



Overall, the RMSEs were found to be inversely proportional to the number of neurons in the middle layer, as was expected. However, the RMSEs of both categories of traffic were found to change with equal proportion and changing the number of neurons did not make malicious traffic stand out.

Figure 4. From 2 to 40 Neurons in the Middle Layer



### Extending to a Deeper Architecture

As shown in Figure 5, adding additional layers to the encoder and decoder sections exhibited the same behavior. Extending to a deeper architecture only compacted the range of error distributions and did not show any significant benefits. Also, adding additional layers has an inherent disadvantage. It exponentially increases the time and space complexity of the model (Srivastava, et al., 2015).

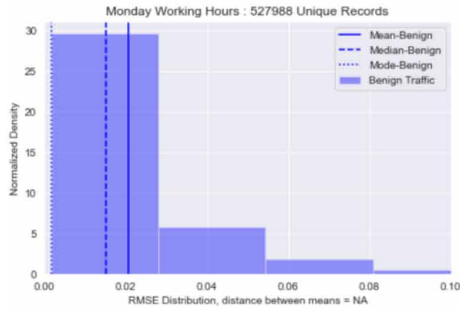
From these observations, it can be concluded that the precise configuration of the autoencoder is not relevant per se (only in this case and with this data). For example, it does not matter if the structure of the autoencoder is (79-9-79), or (79-11-79) instead of (79-10-79) as long as the same structure is consistently used throughout.

For the rest of this study, autoencoders with one hidden layer (79-10-79), the *ReLU* activation function at each layer, and the *Adam* optimizer with default hyper parameters were used.

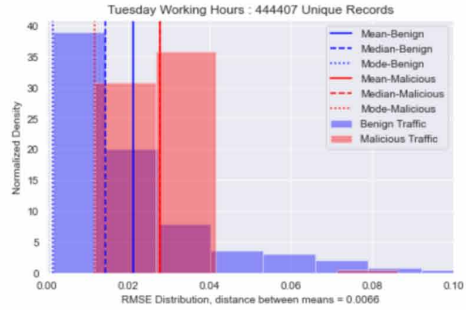
### TIME BASED ANALYSIS

As mentioned in the section on ‘Feature Selection and the curse of Dimensionality’, encoding timestamps to numeric values and including them in the training data can influence any model to classify attacks based on the time slots in which those attacks were carried out. In this experiment, timestamps were used to analyze the trend of autoencoder RMSEs. Most studies in the literature had dropped the *Timestamp* column. This section explores this interesting but often ignored dimension.

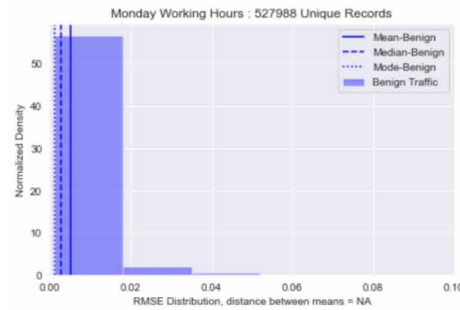
Figure 5. RMSE Distribution with additional layers



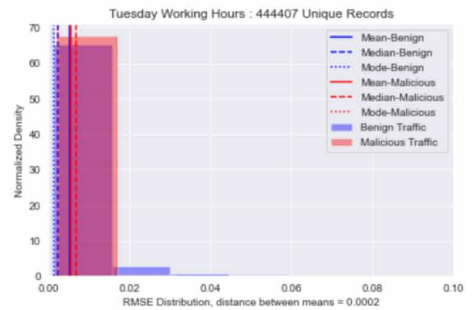
(a) Monday RMSE Distribution with 1 middle layer



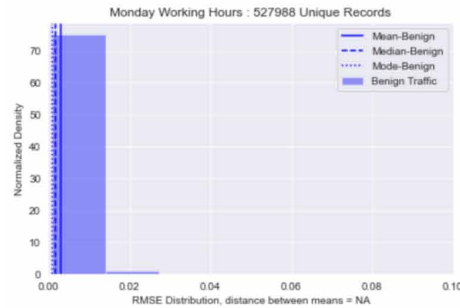
(b) Tuesday RMSE Distribution with 1 middle layer



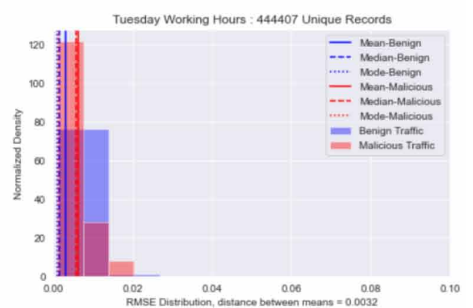
(c) Monday RMSE Distribution with 3 middle layers



(d) Tuesday RMSE Distribution with 3 middle layers



(e) Monday RMSE Distribution with 5 middle layers



(f) Tuesday RMSE Distribution with 5 middle layers

## Data Preparation – Phase 2

The features: ‘Source IP Address’, ‘Destination IP Address’, ‘Flow ID’ and one duplicate instance of ‘Fwd Header Length’ were dropped as explained earlier. Duplicate records were removed from the training data, but this time they were retained in the testing data. ‘Timestamps’ were not included in the input to the autoencoder. They were stored in another variable and were used to visualize the autoencoder RMSEs in chronological order.

### Appending Microseconds to Timestamps

The original timestamps in the dataset are granular to the level of seconds. Since more than one traffic flow can be generated within one second, visualizations based on these timestamps would have resulted in lost details and would have been equivalent to excluding a part of the available data without reasonable justification. An example is highlighted in Figure 6. As a workaround, microseconds were appended to make each timestamp unique, as shown in Figure 7. This workaround ensured that all the records were used in the analysis.

### Encoding timestamps to Datetime objects

The Pandas (McKinney, 2010) (pandas development team, 2020) function `to_datetime()` was used to encode the timestamp strings to datetime objects.

Figure 6. Original Timestamps

Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...
9:01:25	0.286817	0.566667	0.833333	0.401709	0.985294	0.826923	0.0	0.535385	0.861528	...
9:01:27	0.000071	0.066667	0.000000	0.038462	0.000000	0.115385	1.0	0.307692	0.000000	...
9:01:27	0.000007	0.000000	0.055556	0.012821	0.022059	0.115385	1.0	0.307692	0.000000	...
9:01:39	0.045188	0.700000	0.694444	0.376068	0.801471	0.826923	0.0	0.410256	0.841746	...
9:01:39	0.000008	0.000000	0.055556	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	...
9:01:39	0.000010	0.000000	0.027778	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	...
9:06:09	0.019603	0.633333	0.694444	0.376068	0.801471	0.826923	0.0	0.451282	0.864939	...

### Time Corrections

In the original dataset, timestamps after 12:59 PM are incorrectly tagged as AM. To correct this, 12 Hours were added to the time objects starting from 1:00 PM to the rest of the day (around 5:00 PM) in each set.

Figure 7. Timestamps after appending Microseconds

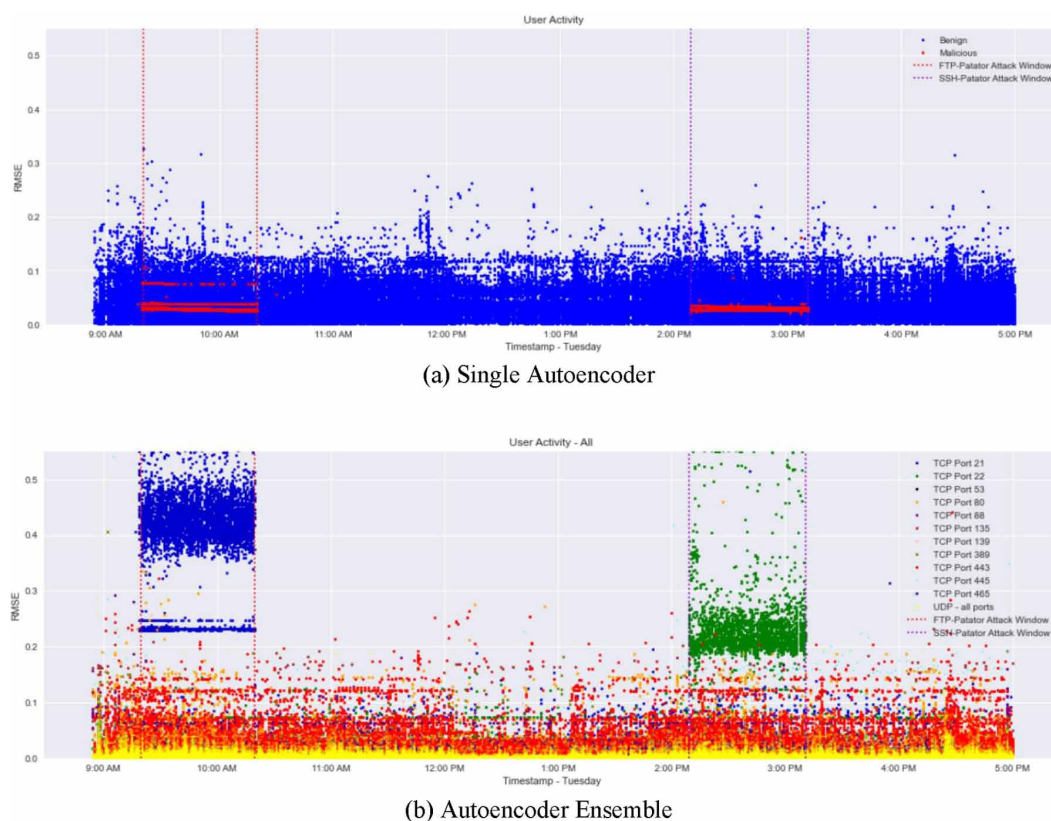
Timestamp	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...
9:01:25:100001	0.286817	0.566667	0.833333	0.401709	0.985294	0.826923	0.0	0.535385	0.861528	...
9:01:27:100002	0.000071	0.066667	0.000000	0.038462	0.000000	0.115385	1.0	0.307692	0.000000	...
9:01:27:100003	0.000007	0.000000	0.055556	0.012821	0.022059	0.115385	1.0	0.307692	0.000000	...
9:01:39:100004	0.045188	0.700000	0.694444	0.376068	0.801471	0.826923	0.0	0.410256	0.841746	...
9:01:39:100005	0.000008	0.000000	0.055556	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	...
9:01:39:100006	0.000010	0.000000	0.027778	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	...
9:06:09:100007	0.019603	0.633333	0.694444	0.376068	0.801471	0.826923	0.0	0.451282	0.864939	...



## Autoencoder Ensemble

The term *ensemble* was inspired by an ensemble of stringed musical instruments where each instrument contributes based on its performing range of musical notes. Using a (79-10-79) autoencoder trained on Monday's data, Tuesday's RMSEs were plotted with respect to time using Matplotlib's (Hunter, 2007) *plot\_date()* method. On Tuesday, brute-force attacks were executed on FTP from 9:20 AM to 10:20 AM, and on SSH from 2:00 PM to 3:00 PM. However, the RMSEs plotted in scatter plot (a) of Figure 8 show that a single autoencoder trained on such a diverse dataset did not give the expected results, and the model was unable to separate brute-force attacks from benign user activity. Extending to a deeper autoencoder architecture did not make brute-force attacks stand out either. Due to space constraints, scatter plots generated by deeper architectures are not shown because they are similar to scatter plot (a) of Figure 8. In the previous section, we explored increasing the complexity of the neural network. In this section, we explored reducing the diversity within the input data. An ensemble of autoencoders was implemented, each trained on a specific substructure within the data. Rather than using a clustering algorithm, destination ports, which are natural clusters were used. Although, it is likely for correlations to exist based on usage (for instance, a script can be used to telnet a remote machine and then initiate an FTP transfer), each service is independent by design. An individual autoencoder was allocated for traffic to each destination port in the well-known port range (0 - 1023). Each autoencoder was independent of the others in the ensemble. The results shown in plot (b) of Figure 8 show that this approach vividly makes brute-force attacks stand out from benign traffic.

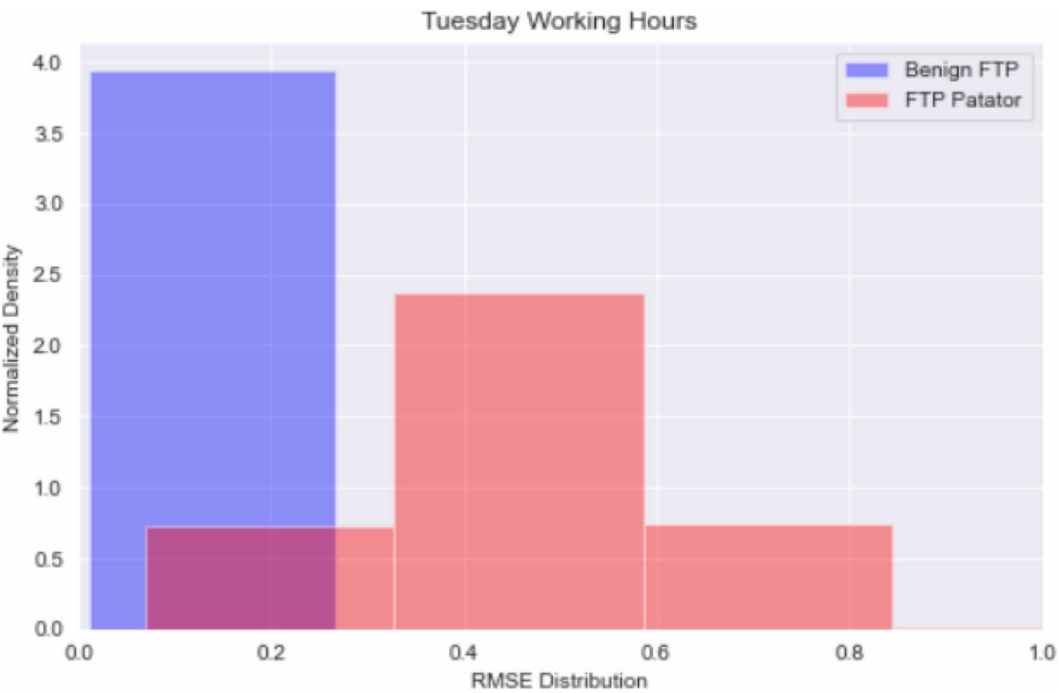
Figure 8. Tuesday RMSEs with respect to time



### Comparison with the Previous Methodology of Classification Based on Thresholds of Reconstruction Errors

Intuitively, it seems apparent that autoencoder ensembles can be equally effective with the methodology of classification based on thresholds of reconstruction errors. However, the histogram of RMSEs from Tuesday's FTP traffic shown in Figure 9 shows that overlaps in reconstruction errors exist irrespective of the diversity within the data. These results further corroborate the theory that building a model to classify each individual observation is not an ideal solution for network intrusion detection.

Figure 9. Tuesday FTP traffic RMSE distribution



### Setup for the Next Set of Experiments

Based on these results, an ensemble of shallow autoencoders, each with three layers (79-10-79) with the *ReLU* function at each layer and the *Adam* optimizer was used in the final setup. Monday's data (without duplicates) was used for training, and the rest of the week's data (with retained duplicates) was used for testing. *MinMaxScaler()* was used to scale the data. Traffic to each destination port in the well-known port range (0 - 1023) was scaled in the range (0-1) and was allocated to an individual autoencoder in the ensemble. Each training set was scaled using the *fit\_transform()* method, and the training parameters were reused to scale the corresponding test set using the *transform()* method. The autoencoder RMSEs were plotted chronologically for each destination port.



## EVALUATION

In this section various categories of malicious activity and their impact on the RMSE distributions were examined. As explained earlier, the RMSEs were plotted with respect to time using Matplotlib's (Hunter, 2007) *plot\_date()* method.

### Brute Force Attacks on FTP and SSH

Brute-force attacks consist of an attacker attempting to guess passwords and gain unauthorized access by trial and error. FTP is a standard protocol used for data transfer. It establishes two connections between the hosts, one for data transfer and the other for control information. Port 21 is used for the control connection and port 20 for the data connection. The FTP server receives authentication over port 21. We therefore analyze traffic to port 21 to identify brute force attacks on FTP. SSH is a network protocol used to remotely access and manage a device. Its predecessor was telnet. The key difference between telnet and SSH is that the latter uses encryption. SSH uses TCP port 22 by default. Patator was the tool used to execute these attacks during the time windows shown in Table 7. Scatter plots (a) and (b) of Figure 10 are representative of activity on TCP ports 21 and 22. Brute force attempts on FTP and SSH noticeably stand out within their respective time-slots in the working hours of Tuesday.

Table 7. Brute Force Attack Windows

Day	Start Time	Stop Time	Malicious Activity
Tuesday	9:20 AM	10:20 AM	FTP-Patator
Tuesday	2:00 PM	3:00 PM	SSH-Patator

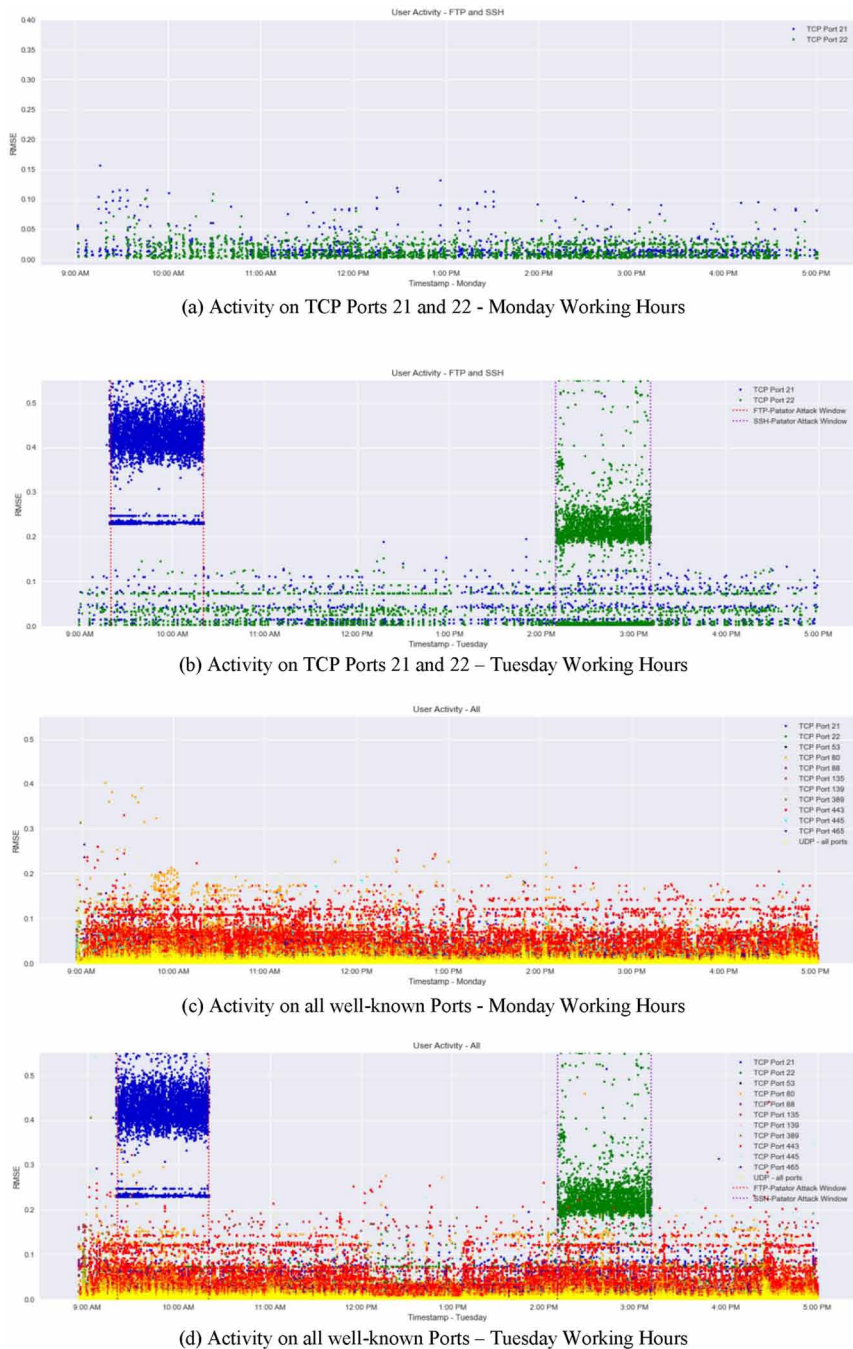
Scatter Plots (c) and (d) of Figure 10 show activity on all ports in the well-known range. In this case, a deviation in FTP and SSH traffic has no effect the other protocols. However, in certain cases, as demonstrated in the section on 'Infiltration and Portscan', malicious activity over one protocol can cause a deviation in the behavior of other protocols. This correlation is based on usage and not by design, as was explained in the section on 'Autoencoder Ensemble'.

### Attacks Over HTTP

Figure 11 is representative of the entire week's activity on TCP Port 80. Table 8 shows the time windows during which each category of attack was executed. Monday's scatter plot shows benign user activity, which is representative of the normal or expected behavior of the HTTP protocol (in this network). Activities on TCP Port 80 from Tuesday through Friday were analyzed to identify datapoints that did not conform to this expected behavior.

On Tuesday, no attacks were executed over HTTP. The scatter plot of Tuesday in Figure 11 therefore looks similar to Monday's plot. On Wednesday morning, various Denial-of-Service (DoS) attacks were executed. These attacks cause unavailability of services by making a server or resource temporarily unavailable (Mantas, et al., 2015). They are typically orchestrated by flooding the target with several illegitimate requests. The overloaded server then places legitimate requests on hold while it is busy processing the malicious requests (Purwanto, 2014). There are several tools available for orchestrating DoS and DDoS attacks. Slowloris and Slowhttptest let a single machine keep connections open with minimal bandwidth to consume the web server resources. In this dataset the attacker was a Kali Linux system, and the victim was an Ubuntu 16.04 system running Apache web server. GoldenEye is a simple tool that attempts to exhaust the resource pools of Hyper Text Transfer Protocol (HTTP) servers. HTTP Unbearable Load King (HULK) is another DoS tool which

Figure 10. Brute Force attacks on FTP and SSH



was originally written for research. It works by generating large volumes of obscure traffic bypassing the cache memory and targeting the web server’s primary resources.

Wednesday’s scatter plot in Figure 11 shows that DoS attack-types HULK and GoldenEye stand out very distinctly in their respective time slots. Slowloris and Slowhttptest do not seem to be apparent

Figure 11. Weekly Activity on TCP Port 80



at first sight, but they can be identified too. In the next section on ‘Modelling RMSEs with respect to time’, some tools for statistically modelling these results are explored to eliminate dependence on visual inspections.

Table 8. Time windows of attacks over HTTP

Day	Start Time	Stop Time	Malicious Activity
Wednesday	9:47 AM	10:10 AM	DoS Slowloris
Wednesday	10:14 AM	10:35 AM	DoS Slowhttptest
Wednesday	10:43 AM	11:00 AM	DoS HULK
Wednesday	11:10 AM	11:23 AM	DoS GoldenEye
Thursday	9:20 AM	10:00 AM	Web attack (brute force)
Thursday	10:15 AM	10:35 AM	Web attack (XSS)
Thursday	10:40 AM	10:42 AM	Web attack (SQL Injection)
Friday	3:56 PM	4:16 PM	Distributed Denial-of-Service - LOIC

On Thursday morning, web-attacks (SQL injection, cross-site-scripting and brute force over HTTP) were executed. As is apparent from Thursday’s scatter plot in Figure 11, our model was unable to identify these attacks. These attack types are discussed under the section on ‘Limitations’.

Distributed Denial-of-Service (DDoS) attacks were executed on Friday. These attacks are similar to DoS attacks, but they involve deployment of multiple systems to launch a coordinated attack. The objective is to flood the bandwidth and/or resources of the victim. The Low Orbit Ion Cannon (LOIC) tool was used to orchestrate these attacks on a target ubuntu machine in the victim’s network (Sharafaldin, et al., 2018). This tool works by sending several illegitimate HTTP GET Requests. As seen in the scatter plot of Friday in Figure 11, these illegitimate requests vividly stand out from genuine requests.

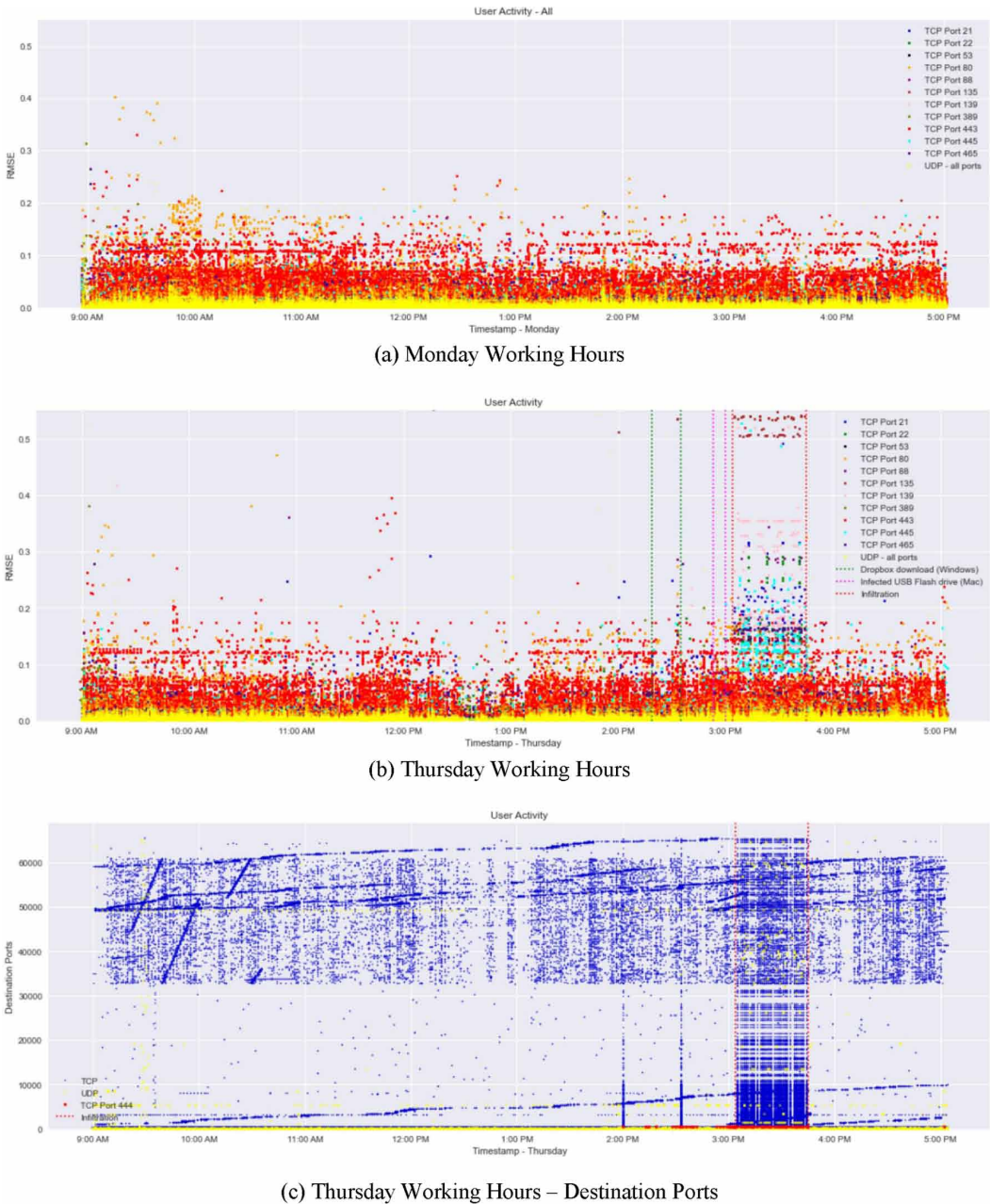
## Infiltration and Portscan

Infiltration is different from the other attacks because it is executed from inside the victim’s network. As described in Table 9, this attack is carried out in two phases. In the first phase, the attacker exploits a vulnerable software in the victim’s network to gain access. In the second phase, the attacker uses the exploited machine to discover more loopholes and conduct various attacks. In this dataset, the first phase began when an infected file was downloaded from dropbox to a windows machine. The file was also copied from a USB flash drive to a Macintosh machine. Usually, in the real world an unsuspecting user inadvertently downloads the infected file. The attacker then uses the backdoor to execute the infected file inside the victim’s network. As shown in plot (b) of Figure 12, traffic generated by the file download on the windows machine resembles benign traffic and cannot be interpreted using this approach. The action of copying an infected file from a USB flash drive to the Macintosh machine occurs locally and is indubitably not expected to be visible in network traffic data. In the second phase, the attacker executed the infected file from inside the victim’s network between 3:04 PM and 3:45 PM on Thursday. This attack changed the behavior of traffic to other well-known TCP ports, as shown in plot (b) of Figure 12. For comparison, Monday’s benign user activity is represented in plot

Table 9. Infiltration attack windows

Day	Start Time	Stop Time	Malicious Activity
Thursday	2:19 PM	2:35 PM	Victim downloads infected file (Windows)
Thursday	2:53 PM	3:00 PM	Infected file is copied from USB flash drive (Mac)
Thursday	3:04 PM	3:45 PM	Attacker executes file using backdoor

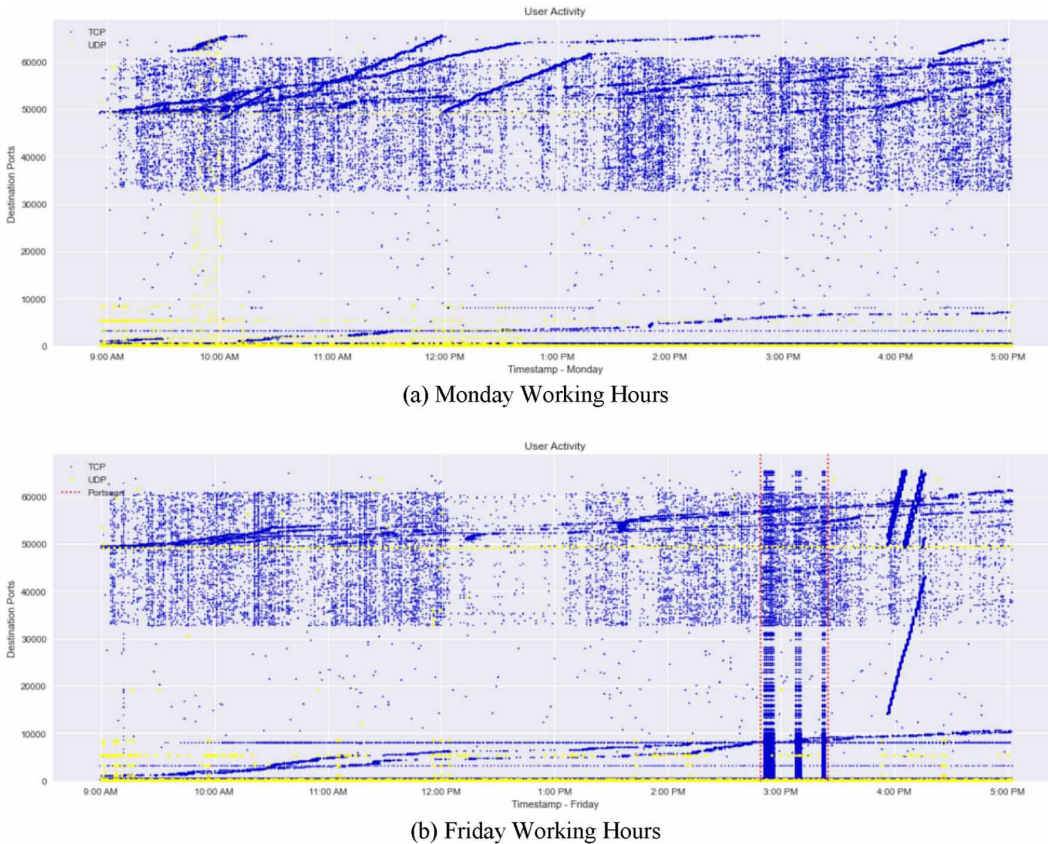
Figure 12. Infiltration



(a) of Figure 12. The attacker also used the backdoor to execute a port scan, which can be identified by a scatter plot of destination ports with respect to time as shown in plot (c) of Figure 12. Another Portscan (unrelated to intrusion) was executed on all the windows machines on Friday afternoon and can be identified using the same methodology as shown in Figure 13.



Figure 13. Portscan



## MODELLING RMSES WITH RESPECT TO TIME

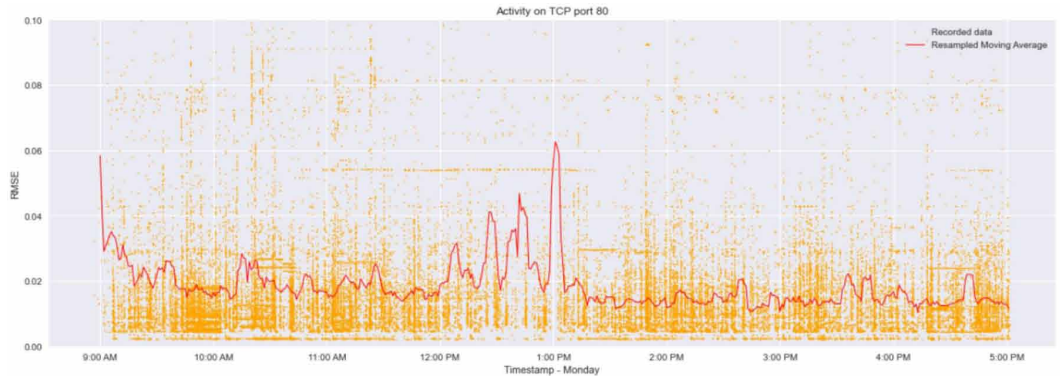
This section explores the likelihood of mathematically modelling the results rather than relying on a visual inspection. HTTP traffic is a suitable candidate for this evaluation because attacks on other protocols had very little or no influence on the RMSEs of HTTP flows (in this dataset). An added advantage is that most attack types in this dataset were orchestrated over the HTTP protocol.

### Data Resampling and Moving Averages

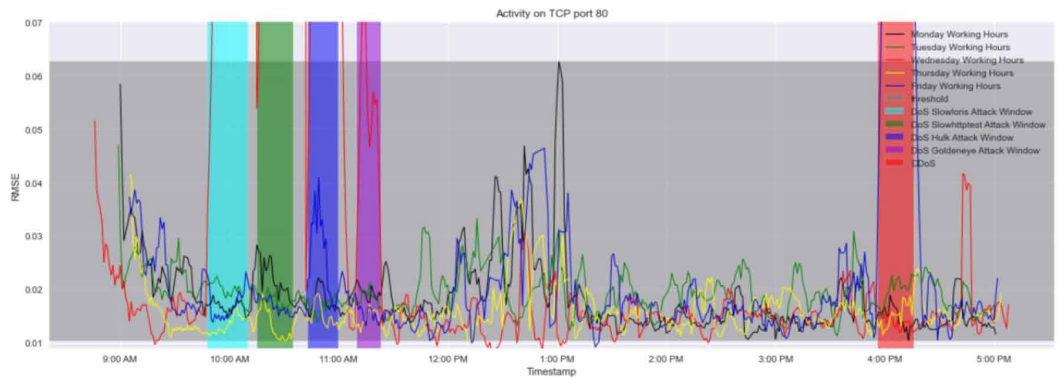
An effective method for smoothing out a curve or filtering out noise is to compute the Moving Average (MA) over a sliding window. MAs can reveal hidden patterns that are not apparent in the original visualization. However, directly applying a sliding window over the recorded RMSEs would have resulted in varying time window lengths. Since the time windows should have constant length, the RMSEs were resampled into bins of 1-minute intervals. The mean of the instances in each bin was the representative value for that bin and was used for the calculation of the Moving Average over a sliding window of 5 minutes. The resulting graph corresponding to the reference benign traffic of Monday is shown in plot (a) of Figure 14. The entire week's activity is shown in plot (b) of Figure 14.

Generically, the resampling rate and sliding window length depend on the data. They are optimized by trial and error and need to be fine-tuned for each network. Choosing a very large window can increase the chances of missing out on certain anomalies (higher False Negatives), while a smaller window can increase the False Positive rate. A resampling rate of 1 minute with a sliding window of

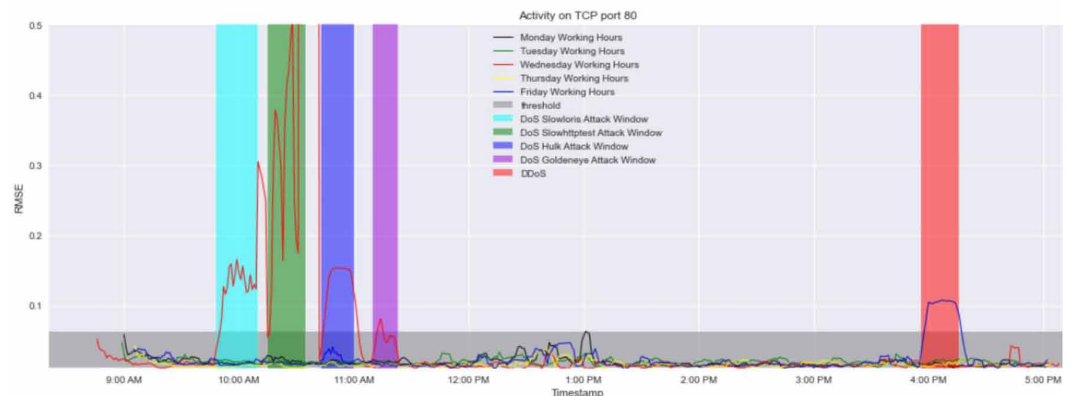
Figure 14. Moving Averages of resampled activity on TCP port 80



(a) Activity on Monday – Resampled Moving Average



(b) Weekly Activity



(c) Weekly Activity – Zoomed out

5 minutes was a perfect fit for this data. The maximum recorded RMSE calculated from Monday's benign data was set as the upper threshold and is the shaded region shown in Figure 14, Figure 15, Figure 16, and Figure 17. In this evaluation, we have used a single threshold for all time windows. However, for some networks it may be prudent to have different thresholds for different time windows depending on the variation in network activity during the course of an average working week.

## Results and Discussion

As shown in plot (b) of Figure 14, the entire week's benign traffic follows a regular pattern. In general, the Moving Averages are higher at the beginning and during the middle of each working day. For the final evaluation, time windows of 5 minutes were taken as individual instances. The results are shown in Table 10. The results are also displayed in Figure 15, Figure 16, and Figure 17 using bar charts for a more intuitive visual interpretation.

Each timestamp is representative of the sliding window spanning the previous 5 minutes (including itself). For example, the instance labelled as 9:05 AM is representative of the network activity from 9:01 AM to 9:05 AM. The upper threshold is the value of the maximum RMSE of the autoencoder when data that is confirmed to be benign is passed through it. The maximum recorded RMSE calculated from Monday's benign data was 0.060369874. When the Moving Average of the RMSE in a specific instance exceeded this threshold, then that instance was flagged as anomalous. For example, on Wednesday, Denial-of-Service attack Slowloris was executed at 9:47 AM. The Moving Average exceeded the threshold in the instance labelled 9:50 AM, thus triggering an alert 3 minutes after the attack was initiated. Similarly, DoS Slowhttptest which orchestrated at 10:15 AM raised an alert at 10:20 AM, Dos Hulk started at 10:43 AM and triggered an alert at 10:45 AM, DoS GoldenEye was initiated at 11:10 AM and triggered an alert at 11:15 AM. On Friday, Distributed Denial-of-Service LOIC which began at 3:56 PM triggered an alert at 4:00 PM. The triggering of an alarm, or the dispatch of a warning email/SMS can thus be automated using an if-else statement on any platform.

## LIMITATIONS

### Limitations of the Methodology

This methodology was successful in identifying FTP/SSH Patator, Infiltration, Portscan, Denial-of-Service and Distributed Denial-of-Service attacks. However, as is apparent from Thursday's scatter plot in Figure 11, the results in Table 10, and the bar charts in Figure 15, this model was unable to detect web attacks (SQL injection, cross-site-scripting and brute force over HTTP) that were carried out on Thursday morning. These activities are best identified at the user level of abstraction within the application layer. It is therefore beneficial to use an application-based IDS in combination with a host-based and/or network-based IDS to effectively identify all categories of malicious activity (Bace & Mell, 2001). Secondly, the training data must always consist of traffic that is confirmed to be benign, which may be challenging to determine in some cases. Also, benign traffic characterizations are likely to change over time as the organization grows and/or adopts newer technologies.

### Limitations Due to Insufficient Data

#### *Heartbleed*

Heartbleed attacks were simulated between 3:12 PM and 3:32 PM on Wednesday over TCP port 444. Monday's data which is our reference benign dataset does not have any traffic to TCP port 444, and therefore the model has nothing to compare. However, the presence of this new, previously unseen traffic is an anomaly in itself.

#### *Botnets*

A python-based tool called Ares was used to simulate a botnet from a machine running Kali Linux in the attack-network (Sharafaldin, et al., 2018). The compromised hosts were five windows systems in the victim-network. Most of the communication occurred over TCP port 8080 on Friday morning between 10:00 AM and 11:00 AM. Although Port 8080 is outside the well-known port range, the organization's stakeholders can choose to include any open port for monitoring. However, the system was unable to successfully model benign behavior of the source and targets because there were very few corresponding traffic flows in the benign reference dataset of Monday.



Table 10 Moving Average of RMSEs at each time delta

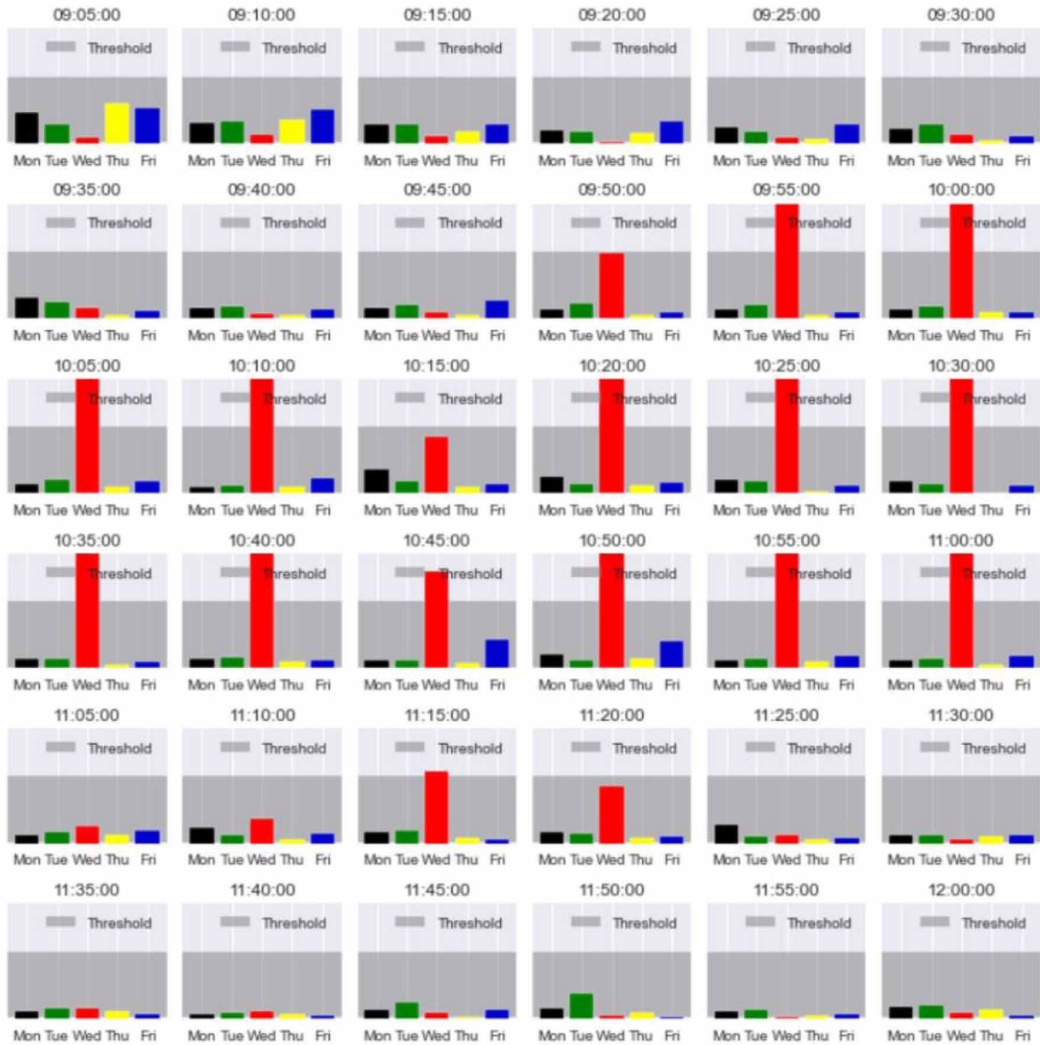
Time stamp	Tue - Moving Average	http Alert	Wed - Moving Average	http Alert	Thu - Moving Average	http Alert	Fri - Moving Average	http Alert	Analysis
9:05:00	0.020847209	No	0.014328677	No	0.041541612	No	0.037371122	No	NA
9:10:00	0.025295857	No	0.01655897	No	0.028180685	No	0.036815131	No	NA
9:15:00	0.023581333	No	0.015536295	No	0.018668727	No	0.02494111	No	NA
9:20:00	0.017015163	No	0.010630559	No	0.017157275	No	0.027812653	No	NA
9:25:00	0.01750865	No	0.01448835	No	0.013538537	No	0.024425144	No	NA
9:30:00	0.023109026	No	0.016562796	No	0.012105407	No	0.016109139	No	NA
9:35:00	0.021421095	No	0.017980863	No	0.012520637	No	0.01544746	No	NA
9:40:00	0.017780809	No	0.013856654	No	0.012525574	No	0.016981501	No	NA
9:45:00	0.017550061	No	0.015130079	No	0.011670313	No	0.023464237	No	NA
9:50:00	0.018428486	No	<b>0.060967029</b>	Yes	0.011644239	No	0.014277122	No	DoS Slowloris
9:55:00	0.018088079	No	<b>0.155330098</b>	Yes	0.012430511	No	0.014287967	No	DoS Slowloris
10:00:00	0.017186532	No	<b>0.147033574</b>	Yes	0.014191753	No	0.014941086	No	DoS Slowloris
10:05:00	0.017940336	No	<b>0.120722249</b>	Yes	0.014814505	No	0.019442427	No	DoS Slowloris
10:10:00	0.014992501	No	<b>0.305027645</b>	Yes	0.014431325	No	0.021597939	No	DoS Slowloris
10:15:00	0.016493992	No	0.053976289	No	0.015108427	No	0.017217883	No	NA
10:20:00	0.014349707	No	<b>0.364110756</b>	Yes	0.016355929	No	0.017913549	No	DoS Slowhttptest
10:25:00	0.017891589	No	<b>0.414629967</b>	Yes	0.011513249	No	0.016421421	No	DoS Slowhttptest
10:30:00	0.016441851	No	<b>0.191375438</b>	Yes	0.010606256	No	0.016287065	No	DoS Slowhttptest
10:35:00	0.014525729	No	<b>2.897972356</b>	Yes	0.013210102	No	0.014576473	No	DoS Slowhttptest
10:40:00	0.016220545	No	<b>2.506264515</b>	Yes	0.015424516	No	0.015893383	No	DoS Slowhttptest
10:45:00	0.015245464	No	<b>0.08750155</b>	Yes	0.013538301	No	0.032527468	No	DoS Hulk
10:50:00	0.014375294	No	<b>0.155219534</b>	Yes	0.017466935	No	0.03208323	No	DoS Hulk
10:55:00	0.017026313	No	<b>0.155316513</b>	Yes	0.014703867	No	0.019356065	No	DoS Hulk
11:00:00	0.016325732	No	<b>0.120197817</b>	Yes	0.012485062	No	0.019518671	No	DoS Hulk
11:05:00	0.016497396	No	0.023118038	No	0.01678113	No	0.019815539	No	NA
11:10:00	0.015704948	No	0.029374764	No	0.013098751	No	0.017716582	No	NA
11:15:00	0.017419385	No	<b>0.06686989</b>	Yes	0.014394598	No	0.013247587	No	DoS GoldenEye
11:20:00	0.013435849	No	0.055037229	No	0.014394598	No	0.015600761	No	NA
11:25:00	0.013767989	No	0.016126773	No	0.013213233	No	0.013703662	No	NA
11:30:00	0.01381433	No	0.013197265	No	0.015281041	No	0.016215749	No	NA
11:35:00	0.016055289	No	0.017669729	No	0.015277145	No	0.013261606	No	NA
11:40:00	0.014669773	No	0.015177098	No	0.013358552	No	0.012704738	No	NA
11:45:00	0.01865842	No	0.014734833	No	0.011071499	No	0.016426482	No	NA
11:50:00	0.027399118	No	0.012013667	No	0.014767911	No	0.01088404	No	NA
11:55:00	0.016699046	No	0.010883489	No	0.012412032	No	0.013872191	No	NA
12:00:00	0.019793156	No	0.01422689	No	0.016667781	No	0.01256546	No	NA
12:05:00	0.017203743	No	0.01339747	No	0.01703911	No	0.029616728	No	NA
12:10:00	0.027799047	No	0.012806246	No	0.012179812	No	0.01677992	No	NA
12:15:00	0.02731358	No	0.018240456	No	0.014284565	No	0.013951379	No	NA
12:20:00	0.02731358	No	0.0114273	No	0.014026377	No	0.014248773	No	NA
12:25:00	0.02731358	No	0.009139874	No	0.020076919	No	0.014248773	No	NA
12:30:00	0.023009137	No	0.012176661	No	0.01290051	No	0.023006035	No	NA
12:35:00	0.018561422	No	0.014449099	No	0.01290051	No	0.038536968	No	NA
12:40:00	0.018561422	No	0.025189552	No	0.036766224	No	0.030140646	No	NA
12:45:00	0.019206095	No	0.025189552	No	0.012136974	No	0.044242744	No	NA
12:50:00	0.018884053	No	0.010763139	No	0.012136974	No	0.044242744	No	NA
12:55:00	0.019620901	No	0.021328012	No	0.017620767	No	0.034350028	No	NA
13:00:00	0.022711594	No	0.011689964	No	0.02028767	No	0.02002678	No	NA
13:05:00	0.014473567	No	0.01006506	No	0.02028767	No	0.030219897	No	NA
13:10:00	0.013524122	No	0.019051494	No	0.010779811	No	0.02185234	No	NA
13:15:00	0.018350596	No	0.015707001	No	0.016778072	No	0.014551365	No	NA

continued on following page

Table 10 . Continued

Time stamp	Tue - Moving Average	http Alert	Wed - Moving Average	http Alert	Thu - Moving Average	http Alert	Fri - Moving Average	http Alert	Analysis
13:20:00	0.018350596	No	0.012409015	No	0.018047935	No	0.011193792	No	NA
13:25:00	0.01298271	No	0.013501127	No	0.014157051	No	0.015688704	No	NA
13:30:00	0.017299762	No	0.01152637	No	0.016666969	No	0.026201335	No	NA
13:35:00	0.01959681	No	0.013839557	No	0.012414099	No	0.013206884	No	NA
13:40:00	0.01963901	No	0.019304	No	0.015354228	No	0.019178543	No	NA
13:45:00	0.016991771	No	0.010901467	No	0.01513657	No	0.017581025	No	NA
13:50:00	0.01510094	No	0.014175772	No	0.015207586	No	0.013098006	No	NA
13:55:00	0.013502764	No	0.015590941	No	0.01790338	No	0.017275409	No	NA
14:00:00	0.016562281	No	0.013687344	No	0.013335975	No	0.013424484	No	NA
14:05:00	0.019211438	No	0.013518545	No	0.013315761	No	0.01366843	No	NA
14:10:00	0.021214669	No	0.016944097	No	0.013284187	No	0.017785488	No	NA
14:15:00	0.017326409	No	0.014779793	No	0.015228897	No	0.022185409	No	NA
14:20:00	0.016465677	No	0.011026433	No	0.015698968	No	0.013487472	No	NA
14:25:00	0.016022472	No	0.012747546	No	0.022304765	No	0.016030065	No	NA
14:30:00	0.01817747	No	0.014807982	No	0.011101968	No	0.014115236	No	NA
14:35:00	0.016245059	No	0.018863539	No	0.015872006	No	0.016996768	No	NA
14:40:00	0.023266713	No	0.015945974	No	0.012024261	No	0.015879084	No	NA
14:45:00	0.022556657	No	0.015154699	No	0.013702124	No	0.014776022	No	NA
14:50:00	0.014645266	No	0.014458937	No	0.017021316	No	0.017327795	No	NA
14:55:00	0.01427266	No	0.017133605	No	0.018956942	No	0.01216101	No	NA
15:00:00	0.01465962	No	0.01503446	No	0.016973293	No	0.015053288	No	NA
15:05:00	0.012460618	No	0.011417894	No	0.019933331	No	0.011340355	No	NA
15:10:00	0.019802635	No	0.015638259	No	0.01872585	No	0.014904481	No	NA
15:15:00	0.020399877	No	0.015009487	No	0.015470637	No	0.011970053	No	NA
15:20:00	0.016783451	No	0.015370065	No	0.014247208	No	0.013771911	No	NA
15:25:00	0.015995054	No	0.012931118	No	0.014020859	No	0.015845037	No	NA
15:30:00	0.016803567	No	0.01320917	No	0.01373286	No	0.01876419	No	NA
15:35:00	0.012929789	No	0.016213767	No	0.017042525	No	0.028122824	No	NA
15:40:00	0.018449361	No	0.022277113	No	0.016702212	No	0.016867416	No	NA
15:45:00	0.025035134	No	0.012893061	No	0.015367035	No	0.023163113	No	NA
15:50:00	0.013329089	No	0.018667849	No	0.013294592	No	0.023114841	No	NA
15:55:00	0.018267723	No	0.015882054	No	0.014060704	No	0.011802883	No	NA
16:00:00	0.014386572	No	0.014782704	No	0.011974471	No	<b>0.103792391</b>	<b>Yes</b>	DDoS LOIC
16:05:00	0.017699595	No	0.013088339	No	0.015112642	No	<b>0.104784445</b>	<b>Yes</b>	DDoS LOIC
16:10:00	0.017667455	No	0.013981654	No	0.018289962	No	<b>0.106361733</b>	<b>Yes</b>	DDoS LOIC
16:15:00	0.021519672	No	0.01206987	No	0.015152539	No	<b>0.094676598</b>	<b>Yes</b>	DDoS LOIC
16:20:00	0.018430018	No	0.013677279	No	0.020633343	No	0.022133552	No	NA
16:25:00	0.018489833	No	0.013872635	No	0.014281971	No	0.022572033	No	NA
16:30:00	0.019598572	No	0.014758112	No	0.016419537	No	0.01387552	No	NA
16:35:00	0.016627805	No	0.013621997	No	0.014221272	No	0.011042435	No	NA
16:40:00	0.014223864	No	0.015548746	No	0.013728461	No	0.016334799	No	NA
16:45:00	0.019103901	No	0.038630103	No	0.016138728	No	0.017543214	No	NA
16:50:00	0.013572844	No	0.014609358	No	0.015736958	No	0.015096923	No	NA
16:55:00	0.014005773	No	0.016282156	No	0.015518796	No	0.015602825	No	NA
17:00:00	0.015394644	No	0.013817556	No	0.013837513	No	0.014285207	No	NA

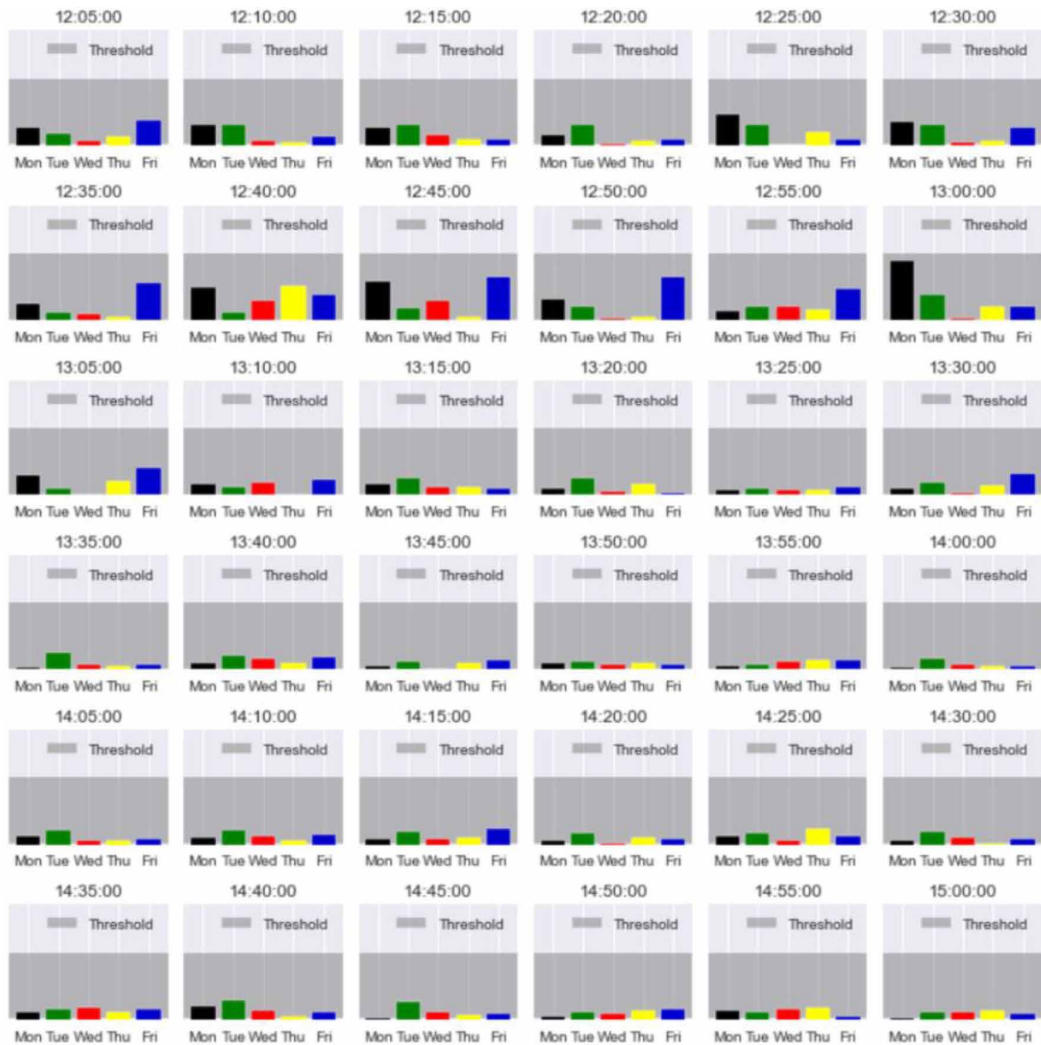
Figure 15. Morning - Moving Average of RMSEs at each time delta



## SUMMARY AND CONCLUSION

This paper has proposed an anomaly-based intrusion detection approach which suggests that it is unnecessary to classify each individual observation. The results support the premise that a subset of network traffic originating due to malicious activity in the application layer is similar in structure to benign traffic when observed in the lower layers and cannot be differentiated without explicit labels regardless of the complexity of the neural network or the diversity within the data. The results also support the hypothesis that timestamps, which occur naturally in network traffic data can be used in conjunction with representation learning to identify deviations from the normal or expected behavior of the network. This can be done by tracking irregularities in autoencoder reconstruction errors that are consistent over a pre-determined time-window. A traffic flow is not necessarily mapped 1:1 to a specific transport connection. A solitary packet can appear benign when analyzed in isolation, but it could be part of a structured denial-of-service attack. Based on these observations and the results of the experiments conducted in this study, it can be concluded that the overlap in autoencoder reconstruction

Figure 16. Afternoon - Moving Average of RMSEs at each time delta

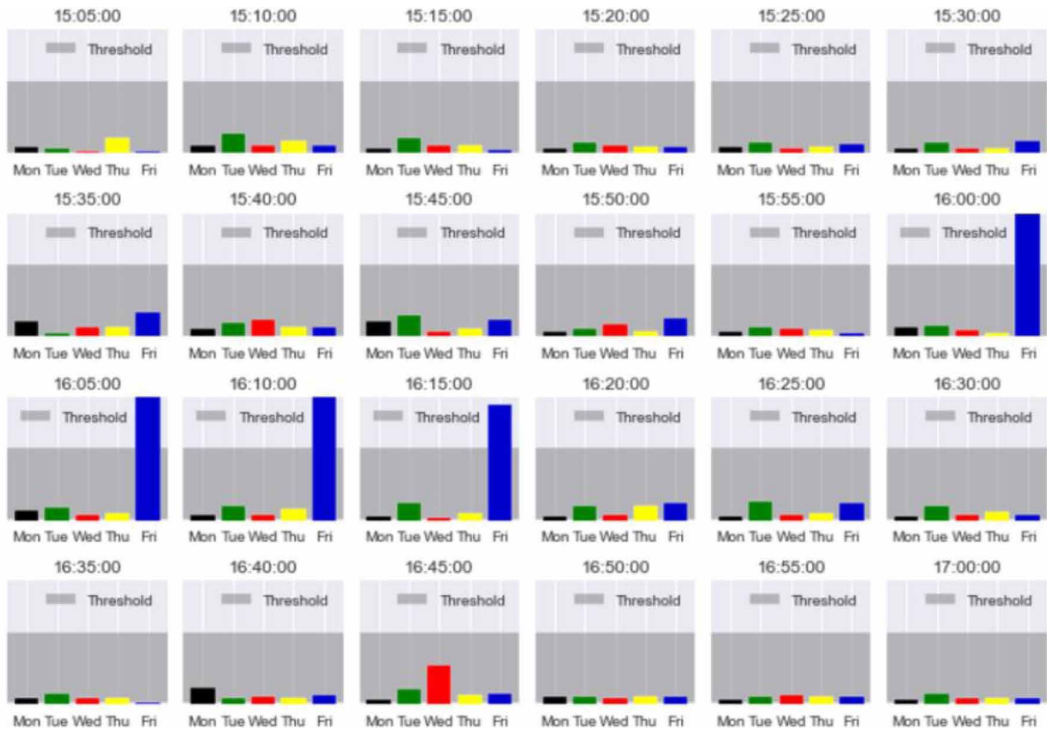


errors of benign and malicious traffic cannot be eliminated but analyzing their distributions with respect to time can provide a different perspective, and it is an important factor to consider while profiling user activity on a network.

## FUTURE DIRECTIONS

The proposed methodology has shown promising results, but its validity is confined to the CICIDS2017 dataset. The solution needs to be evaluated on other datasets and live networks. The data can be pruned for better results. Other metrics like entropy analytics and e-similarity can be explored. Simulated attacks over HTTPS must be tested to provide useful insights about the effectiveness of the methodology on encrypted communication. Data from audit logs in the application layer can be included in the analysis to overcome the limitations mentioned in the previous sections. And finally, a provision can be made to accommodate the phenomenon of data drift, so that the models can be retrained as and when traffic characterizations change naturally with time.

Figure 17. Evening - Moving Average of RMSEs at each time delta



## ACKNOWLEDGMENTS

This research was supported by the University of Cincinnati's Center of Academic Excellence for Cyber Defense (CAE-CD) and the National Security Agency [Grant Number: H98230-17-1-0345]. The material has been revised several times over the past two years in order to articulate as accurately as possible the progression of thoughts and ideas that had evolved through the course of this study. The authors would like to thank their colleagues and friends for peer-reviewing this work, in particular Mr. Amer Al Baidhani (soon to be Dr.). The authors are also thankful to the reviewers at IGI Global for their valuable feedback and suggestions. And, last but not the least, the authors are eternally grateful to Dr. Jess Kropczynski and Ms. Lauren Kirgis from the University of Cincinnati for their consistent support and encouragement throughout the course of this study.

## REFERENCES

- Abadi, M. et al., 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. s.l.:s.n.
- Anbar, M., . . . 2016. Comparative performance analysis of classification algorithms for intrusion detection system. s.l., s.n., p. 282–288.
- Aung, Y. Y., & Min, M. M. 2017. An analysis of random forest algorithm based network intrusion detection system. s.l., s.n., pp. 127–132.
- Bace, R. & Mell, P., 2001. *NIST special publication on intrusion detection systems*, s.l.: s.n.
- Catillo, M., Rak, M., & Villano, U. (2019). Discovery of DoS attacks by the ZED-IDS anomaly detector. *Journal of High Speed Networks*, 25, 349–365.
- Catillo, M., Rak, M., & Villano, U. 2020. 2L-ZED-IDS: A Two-Level Anomaly Detector for Multiple Attack Classes. s.l., s.n., p. 687–696.
- Chandola, V., Banerjee, A. & Kumar, V., 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 7, Volume 41.
- Chang, Y., Li, W., & Yang, Z. 2017. Network Intrusion Detection Based on Random Forest and Support Vector Machine. s.l., s.n., pp. 635–638.
- Choi, H., Kim, M., Lee, G., & Kim, W. (2019). Unsupervised learning approach for network intrusion detection system using autoencoders. *The Journal of Supercomputing*, 75, 5597–5621.
- Chollet, F. & others, 2015. *Keras*. s.l.:s.n.
- Choudhury, S., & Bhowal, A. 2015. Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. s.l., s.n., p. 89–95.
- Claffy, K., Polyzos, G., & Braun, H. W. 1989. Internet traffic flow profiling. In: *Technical Report TR-CS93-328*. s.l.:University of California at San Diego.
- Davis, J. J. & Clark, A. J., 2011. Data preprocessing for anomaly based network intrusion detection: A review. *computers & security*, Volume 30, p. 353–375.
- Dong, B., & Wang, X. 2016. Comparison deep learning method to traditional methods using for network intrusion detection. s.l., s.n., p. 581–585.
- Doyle, A. C., 1859-1930. *The adventures of Sherlock Holmes*. s.l.:Garden City, N.Y.,Doubleday and Co..
- Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I., & Ghorbani, A. A. 2016. Characterization of encrypted and vpn traffic using time-related. s.l., s.n., p. 407–414.
- Farahnakian, F., & Heikkonen, J. 2018. A deep auto-encoder based approach for intrusion detection system. s.l., s.n., p. 178–183.
- Habibi Lashkari, A., 2018. CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. 8.
- Hagan, M. T., Demuth, H. B., & Beale, M. (2014). *Neural network design*. s.l.:Martin T. Hagan and Howard B. Demuth.
- Harris, C. R., Millman, K. J., Stefan, J. & team, 2020. Array programming with NumPy.
- Heaton, J., 2015. *AIFH, Volume 3: Deep Learning and Neural Networks*. s.l.:s.n.
- Heba, F. E., Darwish, A., Hassanien, A. E., & Abraham, A. 2010. Principle components analysis and support vector machine based intrusion detection system. s.l., s.n., p. 363–367.
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9, 90–95.
- Japkowicz, N., Myers, C., Gluck, M., & Associates. 1995. A novelty detection approach to classification. s.l., s.n., pp. 518–523.

- Javaid, A., Niyaz, Q., Sun, W., & Alam, M. 2016. A deep learning approach for network intrusion detection system. s.l., s.n., p. 21–26.
- Kingma, D. P. B. J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal. American Institute of Chemical Engineers*, 37, 233–243.
- Lashkari, A. H., Draper-Gil, G., Mamun, M. S. I., & Ghorbani, A. A. 2017. Characterization of tor traffic using time based features. s.l., s.n., p. 253–262.
- Mantas, G. et al.. (2015). Application layer denial of service attacks taxonomy and survey. *International Journal of Information and Computer Security*, 7, 216–239.
- McKinney, W. 2010. Data Structures for Statistical Computing in Python. s.l., s.n., pp. 56–61.
- Mirsky, Y. D. T. E. Y. S. A. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*.
- Obeidat, I. M. et al.. (2019). Intensive Pre-Processing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques. *International Journal of Interactive Mobile Technologies*, 13, 70–84.
- Panda, M., & Patra, M. R. 2009. s.l., s.n., p. Ensemble of classifiers for detecting network intrusion 510–515.
- pandas development team, T., 2020. pandas-dev/pandas: Pandas. s.l.:Zenodo.
- Patil, R., Dudeja, H., & Modi, C. (2019). Designing an efficient security framework for detecting intrusions in virtual network of cloud computing. *Computers & Security*, 85, 402–422.
- Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, Volume 12, p. 2825–2830.
- Pedregosa, F. et al.. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Purwanto, Y. K. H. R. B. 2014. 2014 8th International Conference on Telecommunication Systems Services and Applications (TSSA). s.l., s.n., pp. 1–6.
- Quittek, J., Zseby, T., Claise, B. & Zander, S., 2004. *Requirements for IP flow information export (IPFIX)*, s.l.: s.n.
- Rajahalme, J., Conta, A., Carpenter, B. & Deering, S., 2004. *RFC3697: IPv6 Flow Label Specification*. s.l.:RFC Editor.
- Rumelhart, D. E., Hinton, G. E. & Williams, R. J., 1985. *Learning internal representations by error propagation*, s.l.: s.n.
- Sakurada, M., & Yairi, T. 2014. Anomaly detection using autoencoders with nonlinear dimensionality reduction. s.l., s.n., p. 4–11.
- Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. s.l., s.n., p. 108–116.
- Shone, N., Ngoc, T. N., Phai, V. D. & Shi, Q., 2018. A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence*, Volume 2, p. 41–50.
- Srivastava, R. K., Greff, K., & Schmidhuber, J. 2015. Training very deep networks. s.l., s.n., p. 2377–2385.
- Tavallae, M., Bagheri, E., Lu, W., & Ghorbani, A. A. 2009. A detailed analysis of the KDD CUP 99 data set. s.l., s.n., p. 1–6.
- Thompson, B. B. et al.. (2002). *Implicit learning in autoencoder novelty assessment*. s.l. IEEE.
- Yousefi-Azar, M., Varadharajan, V., Hamey, L., & Tupakula, U. 2017. Autoencoder-based feature learning for cyber security applications. s.l., s.n., pp. 3854–3861.
- Zavrak, S. İ. M. (2020). *Anomaly-Based Intrusion Detection From Network Flow Features Using Variational Autoencoder*. s.l. IEEE.

*Nitin Mathur received his Master's degree in Information Technology from the University of Cincinnati in Fall 2020. His interests include IT Infrastructure, data centers, virtualization, storage, messaging, cloud computing, and playing the violin.*

*Chengcheng Li received his M.S. and Ph.D. degrees in computer science and obtained an MBA degree. His research and teaching interests are in cybersecurity and data technologies. In 2014, he started the cybersecurity programs at the School of IT of University of Cincinnati that led to UC's designation as a National Center for Academic Excellence in Cyber Defense. Since then, he was awarded more than \$6 Million as the principal investigator in cybersecurity related grants sponsored by the National Science Foundation and the National Security Agency. Dr. Li has over 15 publications primarily in cybersecurity and infrastructure design.*

*Bilal Gonen is an Assistant Professor in the School of Information Technology at the University of Cincinnati. Dr. Gonen has expertise in cloud computing, IT infrastructure, advanced storage technologies, and advanced system administration, Blockchain, Virtualization, AWS, Hyper-V, Machine Learning, Programming in Java, C, C++, Python.*

*Kijung Lee is an assistant professor in the School of Information Technology at the University of Cincinnati. He received a PhD degree in Information Studies from the College of Computing and Informatics at Drexel University. He utilizes quantitative analyses to examine and validate interesting research questions involving communication and information theories, credibility and veracity of information on social media, and user perspective of privacy and security in Online Social Networks.*