Issues in Information Systems
Volume 25, Issue 4, pp. 277-286, 2024

DOI: https://doi.org/10.48009/4_iis_2024_122

Machine Learning Using Autoencoder Method: An Application to Detecting Anomalies in Internet Traffic

Vadim Bashurov, *Comtrade, Serbia, bashurov@mail.ru*
Paul Safonov, *Saint Cloud State University, United States, safonov@stcloudstate.edu*

Abstract

This study presents a method to detect unusual activity in Internet traffic using an autoencoder, a type of neural network. We use the publicly available UNSW-NB15 dataset, which includes network traffic data and labels indicating hacker attacks. The data is processed using an entropy method to prepare it for the autoencoder. Our analysis involves training the autoencoder to reconstruct the input data. By measuring the difference between the original data and the reconstructed data we can identify anomalies. Namely, large differences, or errors, suggest anomalies, which often correspond to malicious activities. This method leverages the autoencoder's ability to learn and represent data efficiently, making it a strong tool for detecting unusual activities, and thus provides a way to enhance network security.

Keywords: machine learning, big data, neural networks, autoencoder, entropy, network security, anomaly detection

Introduction

Any Internet service generates Internet traffic, which can be likened to postal traffic. For example, any letter sent to a post office has attributes like the source address, source postal index, destination address, and destination postal index. Additionally, there are other attributes like letter type, weight, size, and the postal service used (e.g., FedEx). Internet traffic has similar attributes. Each request to an Internet service includes attributes like the source IP address and port (similar to the source address in postal traffic), destination IP address and port, protocol type, request size, etc.

Naturally, Internet services can encounter various issues, including hacker attacks. The rapid growth of network data and the elaborate nature of cyberattacks have urged the need for effective anomaly detection. The first anomaly detection method for intrusion detection was proposed almost 40 years ago (Denning, 1987). Traditional methods often struggle to identify complex and evolving attack patterns. To address this challenge, we propose leveraging the concept of entropy as a measure of uncertainty and variability in network traffic (Baez et al., 2011). Recently, entropy-based methods which rely on network feature distributions has been of a great interest (Berezinski et al., 2015).

In this paper, we further advance the entropy approach, and aim to detect hacker attacks on Internet services using a machine learning (ML) algorithm. These attacks often manifest as anomalies in Internet traffic. For instance, in postal traffic, a very heavy letter might be an anomaly — it could potentially contain a dangerous object. Similarly, we can recognize suspicious mail addresses.

As a particular machine learning method, we will use the autoencoder approach to detect anomalies in a dataset of Internet traffic. Timčenko et al., 2020 used the data clustering method, as well as other machine learning methods were employed in the past (Ranjan et al., 2007).

In our research, we use a public dataset of Internet traffic with regular attributes and anomalies. A good example of such dataset is the UNSW-NB15 available on a public server (Moustafa et al., 2015). We successfully exploited this dataset in our previous research (Bashurov, Safonov, 2023), which applied Shannon entropy (Shannon, 1948) for cyberattacks detection. Another public dataset can be found in Sarhan et al., 2020. This study develops a Python code to demonstrate how to apply the autoencoder algorithm to detect anomalies in the dataset.

Dataset structure

The input dataset has 3,000,000 rows, each containing 49 fields. The dataset is divided into four CSV files, each with 700,000 lines. The Table 1 below shows the structure of the file UNSW-NB15_1.csv. Each row in the table represents a simple Internet request. Together, these requests form the Internet traffic of a server.

Every row in the table has attributes. There are 47 (0…46) regular attributes in each row. These include the source addresses (IP and port), target addresses (IP and port), protocol type, size of request, size of response, etc. Additionally, there are 2 artificial attributes (columns 47 and 48) provided by the dataset owner.

Table 1: Structure of the Dataset (4 rows × 49 columns)

	0	1	2	3	4	47	48
0	59.166.0.0	1390	149.171.126.6	53	udp	NaN	0
1	59.166.0.0	33661	149.171.126.9	1024	udp	NaN	0
2	59.166.0.6	1464	149.171.126.7	53	udp	NaN	0
3	59.166.0.5	3593	149.171.126.5	53	udp	NaN	0

The owner marks every request in the dataset with the label "Type of Hacker Attack" and the corresponding number of the hacker attack. If an Internet request is natural, it has the label "No attack" in column 47 and the digit "0" in column 48. If there was, for example, a DDoS attack, column 47 shows the label "DDoS attack" and column 48 shows the digit "5". Let's modify column 48 to an array of anomaly numbers and print the number of records. The result of the modification is shown in Figure 1.

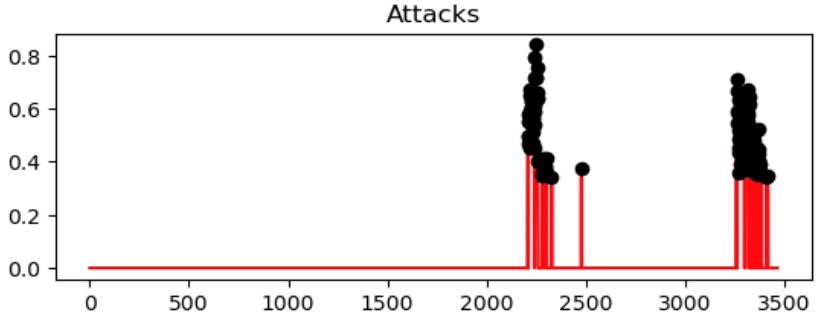

Figure 1: Attacks are marked by black dots.

Entropy method

Next, we prepare our data for the autoencoder algorithm. This autoencoder data must be normalized arrays of numbers in the range [0, 1]. We have two types of attributes: numbers and strings. Numerical data can be easily transformed. Labeled data, such as protocol type, can be converted to numerical data using the entropy method.

Let's describe in more detail how we transform the string array to an array of numbers in the range [0, 1]. We will calculate the entropy of each set of 500 elements from the input array and obtain an array of entropy values.

The entropy H of a sequence can be calculated using the formula:

$$H = -\sum_{i=1}^{n} p_i log_2(p_i)$$

where p_i is the probability of occurrence of each unique value in the sequence.

By applying this method, we convert labeled data into numerical data suitable for the autoencoder algorithm.

Example:

> Consider a protocol array where all elements are the same, e.g., [‘tcp’, ‘tcp’, ‘tcp’, ‘tcp’]. Since there is only one unique element, the probability p_i of this element is 1, and the entropy is 0 since
>
> $H = -(1*log_2(1))=0$.

We calculate entropy for each string attribute using a sliding window with a size N=500 and an overlap of 250 elements between windows. This approach results in 3,500 output points (Figure 2).

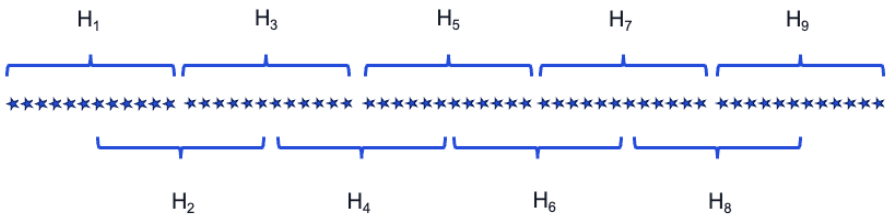

Figure 2: Converting string attribuite * to entropy values.

We calculate entropy for the protocol array (700 000 elements) using a sliding window with a size N=500 and an overlap of 250 elements between windows.

The result of the modification protocol attribute to entropy values Hi is shown in Figure 3.

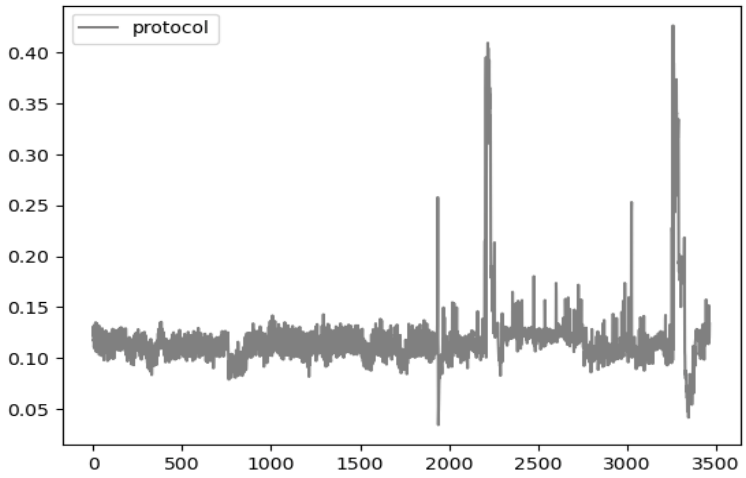

Figure 3: Entropy of protocol attribute.

Calculate entropy for the source ip and port (700 000 elements in each array) using a sliding window with a size N=500 (Figure 4).

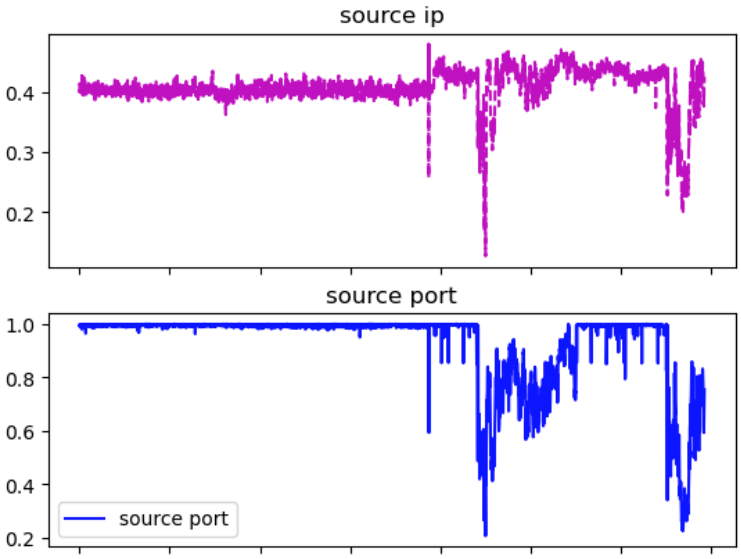

Figure 4: Entropy of source IP and source port attribute.

Calculate entropy for the destination ip and port (700 000 elements in each array) using the same sliding window with a size N=500 (Figure 5).

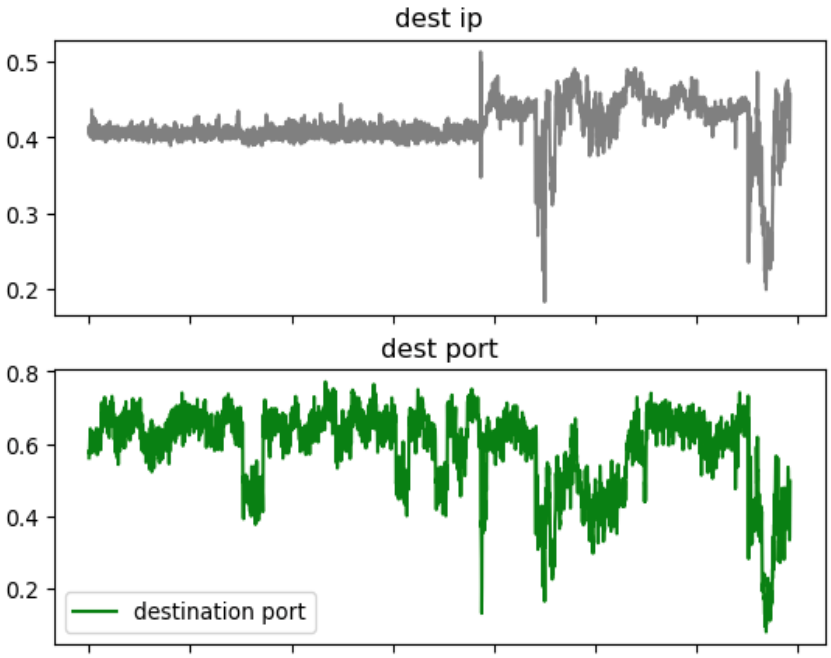

Figure 5: Entropy of destination IP and source port attribute

Autoencoder

An autoencoder is a type of neural network (Figure 6), that learns to compress data into a smaller size (encoding) and then recreate the original data from this compressed form (decoding). It's used for reducing the dimensions of data and for detecting patterns or anomalies.

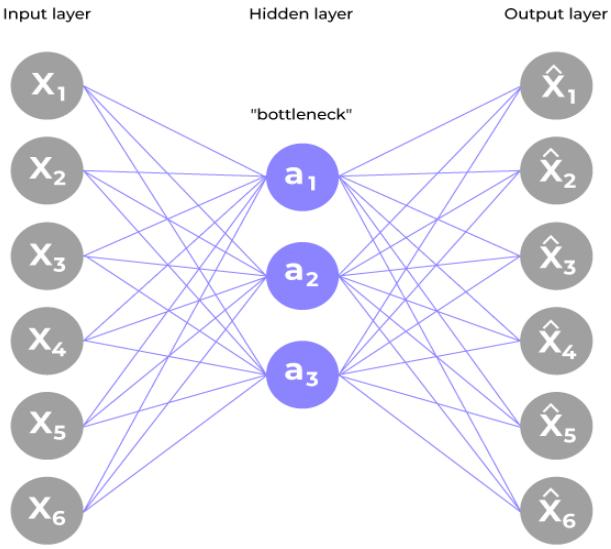

Figure 6: Structure of autoencoder layers.

A typical example of an autoencoder is JPEG compression. In this process, we take an original bitmap image (input layer), encode it to JPEG compressed data (hidden layer), and decode it back to the original format (output layer). The difference between the input and output data represents the anomaly in our case. Let's build the model using Python machine learning libraries. We can use popular libraries such as TensorFlow and Keras for building the autoencoder. Then load the dataset, normalize the numerical data, and transform the string data into numerical values using the entropy method. Then define the structure of the autoencoder, including the input layer, hidden layers, and output layer.

Finally, train the autoencoder using the preprocessed dataset and use the trained model to reconstruct the input data and detect anomalies by comparing the input and output data.

We use Keras libraries to build the neural network. Input data contains the preprocessed attributes data. The input dimension varies from 1 to 40, representing the number of attributes. An encoding layer with 1-100 units and ReLU activation is added. A decoding layer with Sigmoid activation is added. The autoencoder is compiled with the Adam optimizer and mean squared error loss function.

Autoencoder Parameters:

- *Model Structure*: Input shape varies according to the dataset, from 1 to 40.
- *Encoding Dimensions*: Number of neurons in the hidden layer (from 1 to 100).
- *Compiling the Model*: Dense layers with ReLU and Sigmoid activation functions. The optimizer is Adam, and the loss function is mean_squared_error.

Training the Model

The model is trained with the input data for both input and target. Training is performed for 50 epochs with a batch size of 32, shuffling the data, and using 20% for validation.

This process trains the autoencoder to learn a compressed representation of the protocol data, enabling it to detect anomalies by identifying data points that significantly differ from the normal patterns it has learned.

Process trains the autoencoder to learn a compressed representation of the protocol data, enabling it to detect anomalies by identifying data.

Results

In the first numerical experiment, we use the following parameters:

- *Input Dimension*: 1 (just one attribute - protocol)
- *Dimension of hidden layer varied from* 1 to 50. As result of the numerical simulation the optimal value equals 5-10. In this case we use 7.

Next, we prepare our data for the autoencoder algorithm.

The red line in the Figure 7 represents labeled attack records, while the blue line represents the encoded input data and the detected anomaly points.

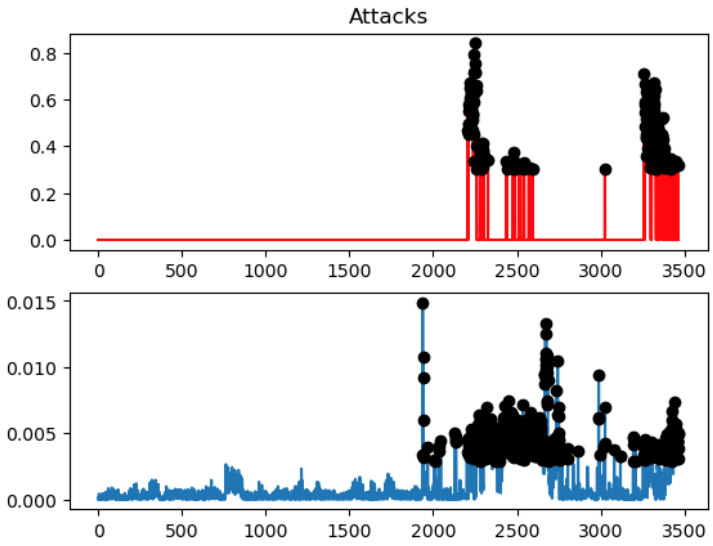

Figure 7: Attacks and detected anomalies for 1 attribute.

In the second numerical experiment, we use the following parameters: input dimension = 2 (protocol, source ip) and hidden dimension = 3. The red line represents labeled attack records. The blue line represents encoded input data and detected anomaly points (Figure 8).

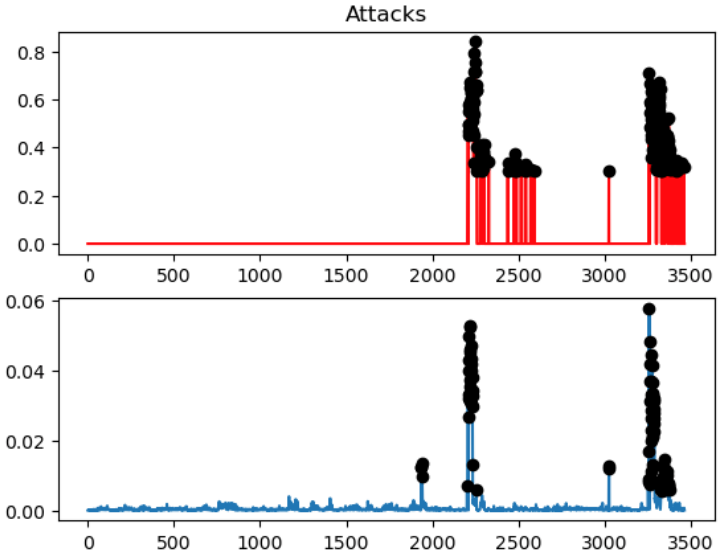

Figure 8: Attacks and detected anomalies for 2 attributes.

In the next numerical experiment, we use the following parameters: input dimension = 3 (protocol, source ip, dest ip) and hidden dimension = 20. The red line represents labeled attack records. The blue line represents encoded input data and detected anomaly points.

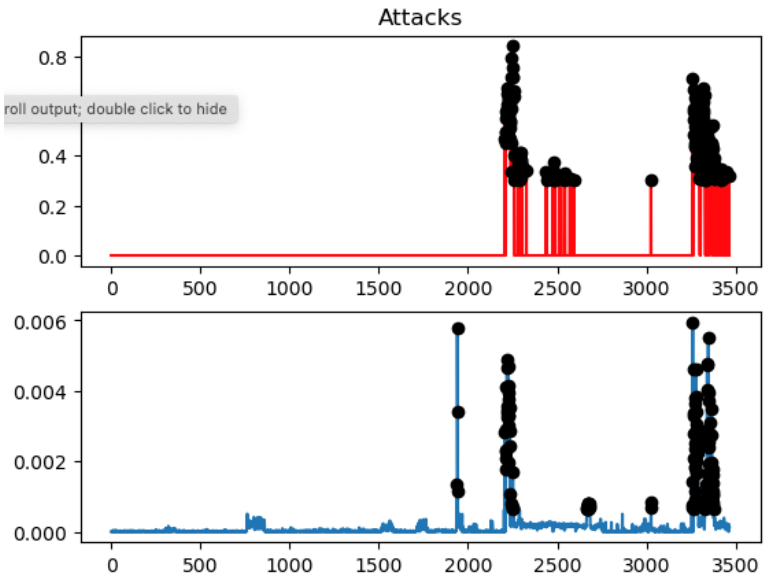

Figure 9: Attacks and detected anomalies for 3 attributes.

Now we have trained the model and applied it to the other three parts of our dataset. In the Figure 10, we can see the detection of anomalies in the second part of the dataset.

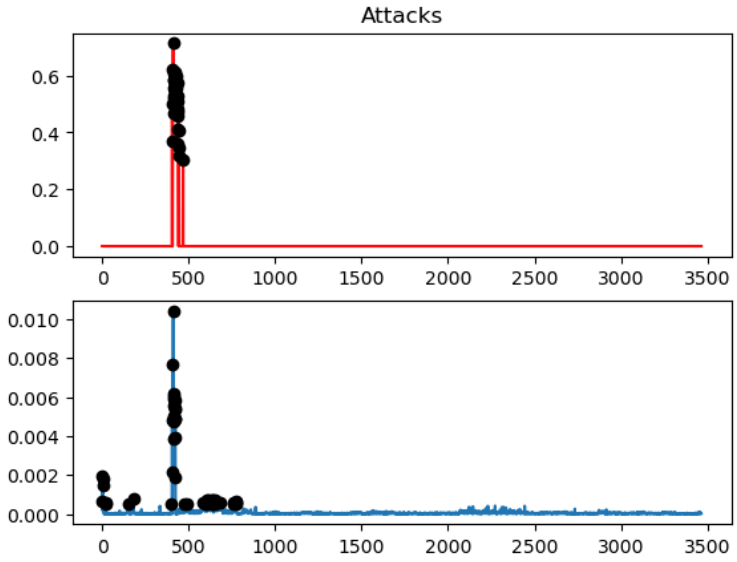

Figure 10: Part 2. Attacks and detected anomalies for 3 attributes.

In the Figure 11, we can see the detection of anomalies in the 3-rd part of the dataset with input dimension = 3 (protocol, source ip, dest ip) and hidden dimension = 20. The red line represents labeled attack records. The blue line represents encoded input data and detected anomaly points.

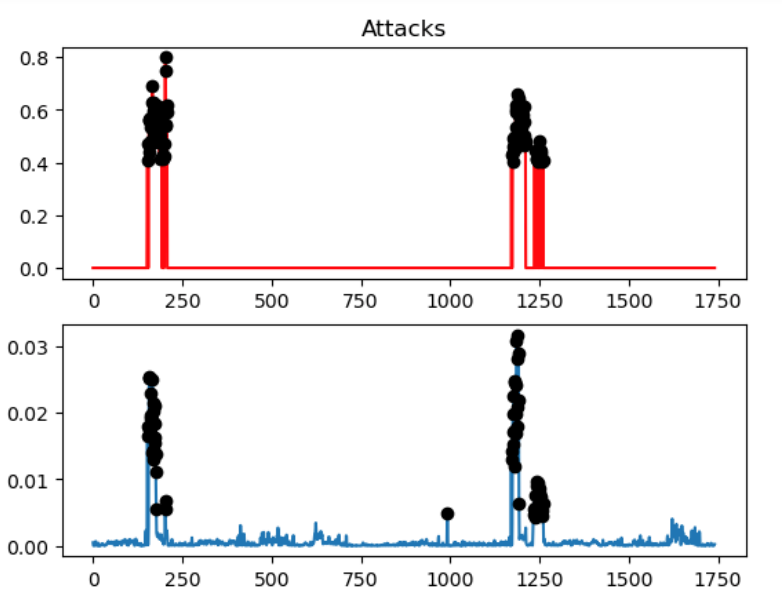

Figure 11: Part 3. Attacks and detected anomalies for 3 attributes.

Discussion

The following should be noted and paid attention to, based on the conducting of this study:

- We should manually transform symbolic and numeric data into shapes suitable for the autoencoder.
- We handle the dimension of input shapes. More shapes do not necessarily yield better results.
- The best attribute must be found manually. Results depend on the size of the encoder layer.
- Better results are typically in the range of [5...20]. The threshold for detecting anomalies can be set manually.

This approach can be applied as an additional service to any server traffic. For that, all we need is a database with two tables, and a Python code for a Telegram Bot:

- Table 1 stores 2,000 parameters of the trained model.
- Table 2 contains the last 500 records of server requests.
- The client then sends a request from their Internet traffic to our service.
- Next, the Telegram Bot sends a push notification if anomalies are detected in the client's traffic.

Some servers exhibit different behavior on regular weekdays compared to weekends. In such cases, we need to build two separate models to account for these variations.

Overall, based on our experiment, the autoencoder method proved its robustness and effectiveness for the anomaly detection. Further research could help compare autoencoder methods performance to other approaches.

References

Baez, J.C., Fritz, T., & Leinster, T. (2011). A Characterization of Entropy in Terms of Information Loss. Entropy, 13, 1945–1957.

Bashurov, V. & Safonov, P. (2023). Anomaly detection in network traffic using entropy-based methods: application to various types of cyberattacks. Issues in Information Systems. 24 (4): 82-94.

Berezinski, P., Jasiul, B., & Szpyrka M. (2015). An entropy-based network anomaly detection method. Entropy 17 (4), 2367–2408.

Denning, D.E. (1987). An intrusion-detection model. IEEE Trans. Softw. Eng., 13, 222–232.

Moustafa, Nour, & Jill Slay. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). Military Communications and Information Systems Conference.

Moustafa, N., Hu, J., & Slay, J. (2019). A holistic review of Network Anomaly Detection Systems: A comprehensive survey. Journal of Network and Computer Applications 128, 33-55.

Ranjan, S., Shah, S., Nucci, A., Munafo, M., Cruz, R., & Muthukrishnan, S. (2007). DoWitcher: Effective Worm Detection and Containment in the Internet Core. Proceedings of 26th IEEE International Conference on Computer Communications, Anchorage, AL, 2541–2545.

Sarhan, M., Layeghy, S., Moustafa, N., & Portmann, M. (2020). NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems. WiCON 2020: In Big Data Technologies and Applications, 117–135.

Shannon, C. E. (1948). A mathematical theory of communication. Bell system technical journal, 27(3), 379-423.

Timčenko, V., Gajin, S. (2020). Time-series entropy data clustering for effective anomaly detection. Proceedings of the 10th International Conference on Information Society and Technology, Information Society of Serbia, ISBN 978-86-85525-24-7, 170-175.