

Web server load prediction and anomaly detection from hypertext transfer protocol logs

Lenka Benova, Ladislav Hudec

Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava, Slovakia

Article Info

Article history:

Received Oct 6, 2022

Revised Feb 13, 2023

Accepted Feb 27, 2023

Keywords:

Anomaly detection

Intrusion detection system

Machine learning

Network traffic prediction

Web server logs

ABSTRACT

As network traffic increases and new intrusions occur, anomaly detection solutions based on machine learning are necessary to detect previously unknown intrusion patterns. Most of the developed models require a labelled dataset, which can be challenging owing to a shortage of publicly available datasets. These datasets are often too small to effectively train machine learning models, which further motivates the use of real unlabeled traffic. By using real traffic, it is possible to more accurately simulate the types of anomalies that might occur in a real-world network and improve the performance of the detection model. We present a method able to predict and categorize anomalies without the aid of a labelled dataset, demonstrating the model's usability while also gathering a dataset from real noisy network traffic. The proposed long short-term memory (LSTM) based intrusion detection system was tested in a real-world setting of an antivirus company and was successful in detecting various intrusions using 5-minute windowing over both the predicted and real update curves thereby demonstrating its usefulness. Our contribution was the development of a robust model generally applicable to any hypertext transfer protocol (HTTP) traffic with almost real-time anomaly detection, while also outperforming earlier studies in terms of prediction accuracy.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Lenka Benova

Faculty of Informatics and Information Technologies, Slovak University of Technology

Ilkovicova 2, Bratislava, Slovakia

Email: lenka.benova@stuba.sk

1. INTRODUCTION

The volume of dangerous network traffic is continuously increasing. There is a significant demand for autonomous data processing and intrusion detection systems based on machine-learning approaches that can detect threats before they become visible. Because a high level of skill is required to comprehend each log entry from a web server request, and even more so to understand sequences of operations such as continuous web requests, there is a solid push to design systems that can identify both known and unknown intrusions.

With an increasing amount of network traffic, the number of anomalies caused by various network misconfigurations continue to grow, resulting in more successful network attacks. Network anomalies must be detected and diagnosed to ensure the confidentiality, availability, and integrity of computer systems; as such, intrusions drain resources and bandwidth, rendering network services unavailable. Crucial computer systems are constantly under attack by numerous attackers on the internet, and intrusion detection systems (IDSs) play an essential role in defending them. Signature-based approaches are used to address attacks by extracting key characteristics and creating a unique signature for an attack. These approaches are highly

effective in combating previously captured attacks. However, they lack the ability to detect new intrusions or zero-day attacks and are not suitable for real-time anomaly detection across large amounts of data [1], [2].

Network intrusion detection systems (NIDSs) are security systems that monitor malicious activity in network traffic and generate alerts when any suspicious activity is detected to further investigate the cause of the alert and take action. Owing to advancements in technology, traditional approaches to network anomaly detection are becoming ineffective as network attacks become more sophisticated [3], [4]. Network anomaly detection is predicated on locating data that do not follow regular behavioural patterns. Despite the availability of numerous methodologies, numerous research hurdles remain. There is no universally applicable anomaly detection approach, as data contains noise, which is an abnormality in and of itself, making it difficult to distinguish. Because intruders are aware of current techniques, there is a dearth of publicly available labelled datasets and the need for more complex and newer techniques.

Various anomaly detection approaches have been developed, but most have limitations when used in real-world situations. This study aims to demonstrate the feasibility of utilising machine learning to detect anomalies and classify metadata in selected types of application servers that offer modular updates to consumers worldwide, with a focus on automation and usability in a real-world setting. Our idea was not to present a new deep learning method; instead, we wanted to show how to get close to real-time anomaly detection from unlabelled datasets comprised entirely of hypertext transfer protocol (HTTP) logs.

2. RELATED WORK

Over the past few years, various anomaly detection methods that incorporate several fields of machine learning have been proposed. Bayesian networks (BNs) have been widely used for grouping problems. Detecting anomalies using BNs has distinct and complementary strengths in spotting anomalies [5], [6]. Nie *et al.* [7] investigated the problem of network traffic modelling. To track flow trends, they proposed using a BN. To evaluate the performance of their model, they collected traffic datasets from Abilene and GÉANT networks. Compared to the three leading methodologies, their solutions regularly surpass them in terms of estimating inaccuracy.

Dynamic BNs were extended by Pauwels and Calders [8] to produce a novel model that allows a better description of the structure and attributes of a log file. To address the drawbacks of regular dynamic BNs, several aspects have been added. For example, functional dependencies were added to provide a better description of the log file structure. They then detailed their approach to generating models that reflected the multidimensional and sequential nature of the log data. Their method performed well in a variety of contexts with varying levels of anomalies in both the training and test datasets, with an area under the ROC curve (AUC) of 0.84 on the BPIC dataset.

A feed-forward neural network was introduced by Poojitha *et al.* [9], [10] to detect anomalies using 10% of the KDD Cup 99 data encompassing both traffic during normal and abnormal behaviour, trained by a backpropagation method. The test results showed that the proposed approach is effective at accurately (94.93% accuracy) detecting various attacks with a low percentage of false positives and negatives. The proposed method detects normal traffic, disk operating system (DOS), and probe attacks well but fails to detect R2L and U2R attacks owing to the very large dimensions of the input data.

Anomaly detection has also been performed with autoencoders (AEs), although this time to identify outliers in the first place. In recent years, AEs have become increasingly popular. AEs have been used in several investigations to detect anomalies. The anomaly detection performance of AE, denoising autoencoder (DAE), principal component analysis (PCA), and kernel PCA approaches were examined by Sakurada and Yairi [11]. A hybrid method with kernel density estimation was utilised in a study by Cao *et al.* [12], and successful results were produced on the KDD dataset.

Recurrent neural networks (RNN), such as long short-term memory (LSTM) or gated recurrent unit (GRU), have been widely utilised in traffic prediction, with excellent results when traffic varies during the week or day [13]–[15], and are thus suitable for network traffic predictions. Lu and Yang [16] used wavelet transformations and an LSTM network to forecast network traffic originating in a domain name system (DNS) server. The purpose was to estimate the mean rate of arriving packets per minute on the following day using the number of arrived packets per second. They built four prediction models: least squares support vector machine (LSSVM), backpropagation (BP) neural network, Linan network, and LSTM network utilising the db3 wavelet, with the proposed model based on wavelet transformation and LSTM network to achieve the best results.

Using an LSTM network, Trinh *et al.* [17] investigated the efficiency of RNN on mobile traffic data. The LSTM network was able to capture the temporal correlation of traffic even for remote timeslots, which made it particularly helpful in real-time applications. Zhuo *et al.* [18] used three network traffic datasets to test their LSTM using a deep neural network (DNN) prediction model. The first dataset includes network

traffic history data from 11 European cities provided by network service providers. The second dataset was based on web traffic history data from the United Kingdom Education and Research Networking Association (UKERNA) academic research website. The third dataset is based on network traffic statistics collected by the Beijing University of Posts and Telecommunications, which is the backbone node of China's education network. Their research revealed that LSTM can be used effectively as a timing sequence forecast model and that adding auto-correlation features improved accuracy when working with large granularity datasets.

Naseer *et al.* [19] used multiple DNN architectures such as convolutional neural networks, AEs, and RNN to propose, implement, and train intrusion detection models. These deep models were trained on the NSLKDD training dataset and tested on NSLKDD's NSLKDDTest+ and NSLKDDTest21 test datasets. The authors implemented traditional machine learning intrusion detection system models with a variety of well-known classification approaches, including extreme learning machine (ELM), k-nearest neighbors algorithm (k-NN), decision tree (DT), random forest (RF), support vector machine (SVM) and naive Bayes (NB). The RoC curve, area under the RoC, precision-recall curve, mean average (mAP) precision and classification accuracy of both DNNs and conventional machine learning models were tested using well-known classification measures. On the test dataset, both deep convolutional neural network (CNN) and LSTM models performed well, with an accuracy of 85 and 89%, respectively.

Malaiya *et al.* [20] conducted an empirical evaluation of deep learning to investigate if it can be used to discover network anomalies. The fully connected network (FCN), variational autoEncoder (VAE), and LSTM with sequence to sequence (LSTM Seq2Seq) structures are used to create a collection of deep learning models. The authors used publicly available traffic data sets from NSLKDD and Kyoto-Honeypot to examine the models. Their experimental results are noteworthy, as the model based on the LSTM Seq2Seq structure performed well on both traffic data sets, with 99% binary classification accuracy.

Kim and Co [21] proved the usefulness of LSTM and CNNs for the task of web traffic anomaly detection by comparing their suggested model to various machine learning methods and achieving superior results. The authors presented a C-LSTM architecture, which was found using parametric tests, model comparison experiments, and data analysis. They employed the C-LSTM to extract patterns from web traffic data that included spatial and temporal information. The characteristics of normal and abnormal data categorized by the C-LSTM were revealed by a confusion matrix and t-SNE analysis. The proposed C-LSTM model classifies and extracts characteristics that could not be extracted using traditional machine learning approaches. Their approach outperforms other cutting-edge machine learning techniques on Yahoo's well-known Webscope S5 dataset, reaching an overall accuracy of 98.6% and recall of 89.7% on the test dataset.

3. DATASET

This study was conducted on a dataset acquired from an antivirus company offering anti-virus and firewall software. The company's application servers provide virus signatures and software module updates to consumers worldwide. Client software downloads these updates at a time set by the client. Consequently, the resource use of these servers varies dramatically over time. While some usage patterns are predictable, such as recently issued virus signature updates, the usual release patterns of various client software modules, day/night cycles across different time zones, and less traffic on vacations and weekends, others can be sudden and difficult to predict.

These updates are tracked by the Zabbix monitoring system and NGINX logs client connections requesting updates. Table 1 lists the currently logged attributes. A dataset containing NGINX logs from numerous update servers in the same hosting area was gathered for this experiment. Thousands of requests are generated each minute and millions each day from single server traffic, creating approximately 50 GB of logs per day. Due to the voluminosity of the data, logs were grouped by minutes and aggregated according to the timestamp when the request was submitted, as shown in Table 2.

This dataset was used to identify potential anomalies, such as a specific anomaly linked to other issues such as client misconfiguration or a rapid connection peak, which could lead to a problem and help us address it early in the process. It is worth noting that the data were noisy and already contained anomalies, which we could not extract because it would require a long and complex expert examination, which would be nearly impossible with such a vast amount of data. We attempted to address this problem by collecting additional data such that anomalies would only account for a small percentage of the total data, and the remainder would be normal, desirable traffic. The timestamp was a critical feature to include in the dataset because collected logs alter their behaviour over days and weeks as clients download less on weekends and at different times during the day. This is why it was split into many features for the model to be able to learn these cycles.

Table 1. Logged attributes

Attribute	Description
remote addr	client IP address
remote user	authentication
http x eset updateid	license key
time local	local time in the Common Log Format
http host	HTTP server host
request method	HTTP request method
uri	path to the update being requested
server protocol	server protocol version
status	response status
body bytes sent	request body length
request length	request length including request line, header, and request body
request time	request processing time in seconds with a millisecond's resolution
http user agent	identification of the client originating the request

Table 2. Logs grouped by time of origin by minutes

Server	Time local	Count	Body bytes sent	Avg body bytes sent	Request length	Avg request length
1	23/Aug/2020:00:00:00	37607	1388254369	36914	23273524	618
1	23/Aug/2020:00:01:00	34044	1159650997	34063	21327793	626
1	23/Aug/2020:00:02:00	34578	1570963146	45432	22320593	645
1	23/Aug/2020:00:03:00	35372	1302698163	36828	22967956	649
1	23/Aug/2020:00:04:00	33731	1345533685	39890	21899394	649

The logs in the dataset ranged from November to February and were collected from numerous servers in the same area with similar update curves and attributes. We chose to use data from servers located in Bratislava and Vienna; a total of five servers. The server is identified by the number above the rows (number one in the sample shown in the Table 3). Each row has columns with values from all five servers, as illustrated in Table 3, with only the first server's data presented. Multiple server lines were concatenated based on the timestamp, resulting in a single line containing data from all parsed servers at the same time (each row represents one timestamp with 40 attributes-8 for each server, as seen in Table 3).

The "count" feature was our target for the anticipated output because we wanted to predict future request count. We were able to forecast future values by shifting the target feature value back in time, according to the chosen prediction window. The data were split at a 90:10 ratio. We aimed for a scaler that could scale the data according to our user-defined parameters because the data from our update servers reached varying maximum and minimum values over time. As a result, we avoided employing library scalers that scale the data based on the minimum and maximum values in the current vector. The own scaler was necessary to scale the data and inverse-scale the data using the same values, resulting in the same scaled values at each time.

Table 3. Input dataset sample showing data from first chosen server

1	count	body bytes sent	avg body bytes sent	request length	avg request length	day	hour	minute
	37607	1388254369	36914	23273524	618	6	0	0
	34044	1159650997	34063	21327793	626	6	0	1
	34578	1570963146	45432	22320593	645	6	0	2
	35372	1302698163	36828	22967956	649	6	0	3
	33731	1345533685	39890	21899394	649	6	0	4

4. NETWORK ARCHITECTURE

Because the update curve is based on previous values, an RNN was chosen to forecast future update server requests. It was possible to determine whether there was a problem with the updates by comparing the actual values to the predicted values using the predicted update curve. The network architecture was chosen according to related work and according to the best prediction results in our experiments comparing GRU and LSTM with various configurations, as shown in Table 4.

An LSTM neural network model was chosen based on processed and aggregated update server logs. It can forecast the future update curve based on a variable-length input. This model can learn and predict common release patterns for various client software modules in the future.

The model was trained by early stopping to avoid overfitting. Each batch consisted of 1,440 timestamps, representing one day. In our situation, the historical traffic patterns throughout workdays,

weekends, and holidays can be tracked by LSTM. To lessen the influence of unusual traffic on the learning curve, we acquired a large dataset of 489 600 records collected over three months. From the total amount of data provided to the network, the anomaly data made up a very minor amount. Naturally, under the professional assumption that the large majority of data is regular traffic free of anomalies.

For the final implementation, we chose the network that performed the best in our experiments in Table 4. It consists of a 512-unit LSTM layer and a dense layer that outputs the target value-request count. The dense layer is used to transform the output of the LSTM layer into a form that can be used to make predictions. It is connected to all units in the preceding and subsequent layers, allowing it to learn a rich representation of the data and make more accurate predictions. The rectified linear unit (ReLU) activation function is used to introduce non-linearity into the network, allowing it to learn more complex patterns in the data. The mean squared error (MSE) loss function and RMSprop optimizer are used to train the model, while early stopping is used to prevent overfitting.

The train and validation split for the network was 90:10. The MinMaxScaler was used to scale the data, as it is non-distorting meaning it preserves the shape of the original distribution. We trained the model on 20 epochs each consisting of 100 steps. The model’s architecture can be seen in Figures 1 and 2.

Table 4. RNN comparison (average loss from 5 runs)

Type	Units	Optimizer	Activation	Loss on the train set	Loss on the test set
GRU	512	RMSprop	sigmoid	0.0040	0.0041
LSTM	512	RMSprop	sigmoid	0.0039	0.0039
GRU	512	Adam	sigmoid	0.0041	0.0042
LSTM	512	Adam	sigmoid	0.0042	0.0043
GRU	512	RMSprop	ReLU	0.0046	0.0047
LSTM	512	RMSprop	ReLU	0.0038	0.0038
LSTM	256	RMSprop	ReLU	0.0054	0.0054
LSTM	1024	RMSprop	ReLU	0.0295	0.0295

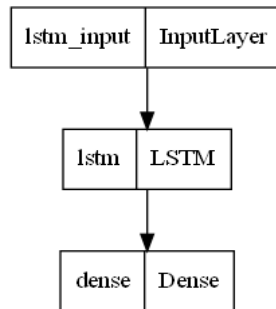


Figure 1. Model’s architecture

```

Model: "sequential"
-----
Layer (type)                Output Shape                Param #
-----
Istm (LSTM)                  (None, None, 512)          1122304
dense (Dense)                (None, None, 1)            513
-----
Total params: 1,122,817
Trainable params: 1,122,817
Non-trainable params: 0
-----
    
```

Figure 2. Model’s description

The update curve prediction trained on one week of data from five servers achieved an MSE loss of 0.00399 on the training set and an MSE loss of 0.0040 on the test set, but it did not represent the peaks correctly enough, as shown in Figure 3. The prediction trained on two weeks of data achieved an MSE loss of 0.0015 on the training set and the same loss on the test set; however, it still did not represent the peaks

correctly enough, despite being better than using 1-week data. For the final training, we used 3 months data and the model's performance can be seen in Figure 4. From this data, the model was able to deduce update patterns.

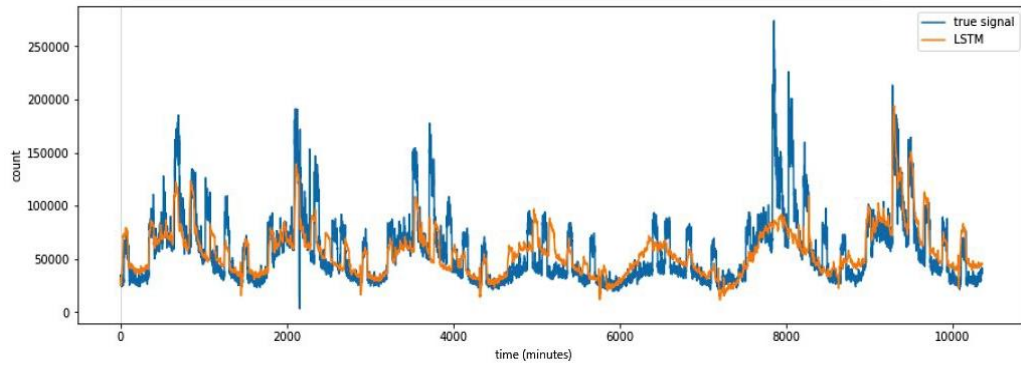


Figure 3. Prediction on 1-week train data

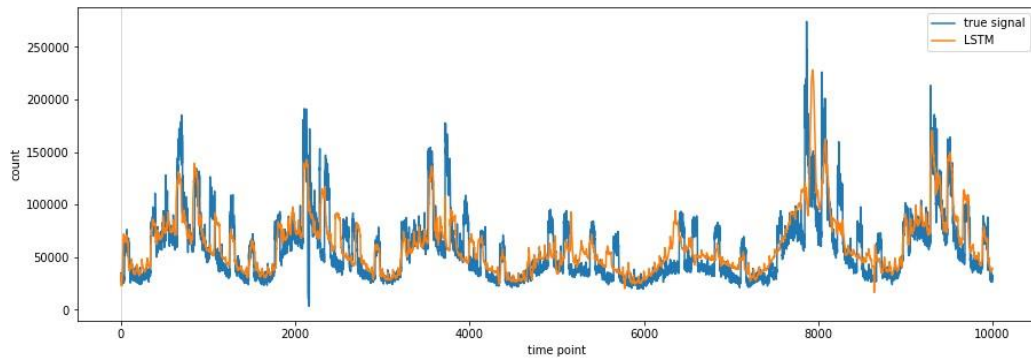


Figure 4. Prediction on train data on three months data

5. CLASSIFICATION

For test data, the model obtained an MSE loss of 0.0021 with a mean absolute error (MAE) of 0.0305 using data representing 489 600 records collected over three months. Figure 4 depicts the prediction on a sample of 10,000 training records, whereas Figure 5 depicts the prediction on a sample of 10,000 test records. Although the loss was higher after two weeks of training, the model was still able to predict the update peaks, which was critical for our future anomaly classification.

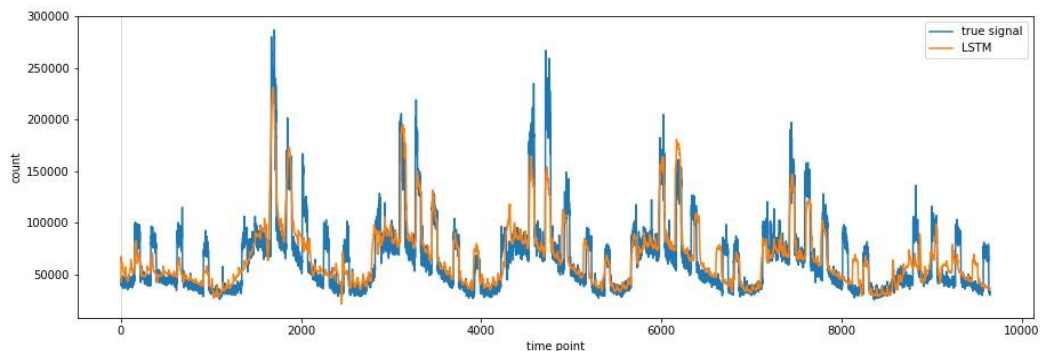


Figure 5. Prediction on test data on three months data

The rules for distinguishing between the anticipated update curve and real traffic were developed based on the analysis of historical data and network behavior. The system was designed to detect anomalies such as sudden spikes or drops in network activity, as well as unusual patterns in traffic that deviated significantly from the expected update curve. By detecting these anomalies, the system could alert network administrators to potential issues and allow them to take corrective actions before they impacted network performance or caused downtime. This capability of the system ensured smooth network operations and reduced the risk of business interruptions.

- Missed update: the anticipated load, which typically lasts several hours, does not occur.
- Unusually large update: the update was not served within an hour because the load did not sink. Clients may not have received everything yet. The load persisted for several hours.
- Unexpected traffic: clients began downloading without us expecting it, indicating a higher load. Someone may have released an additional update, or it may be New Year’s Day or the day after the holidays.
- Degraded performance: the amount of traffic is substantially lower than expected.

The actual and predicted update curves were both windowed using a 5-minute window for classification, and the trends of both curves were observed. Unexpected traffic was defined as an increasing trend in the actual update curve but a decreasing trend in the prediction. It was classified as a missed update if the actual update curve had a descending trend but the prediction showed an ascending trend. The remaining two groups were determined using a threshold based on the window’s maximum value if the trends match. A degraded performance occurs when the maximum value of the actual curve is less than the maximum value of the predicted curve by a value greater than or equal to the threshold. An update is classified as an unusually large update if the maximum value of the actual curve exceeds the predicted curve’s maximum value by a value greater than or equal to the threshold.

The actual and predicted curves for one day are depicted in detail in Figure 6. At the peak, where the selected time point is located, the system generates three notifications. The curve was initially classified as an unexpected update because the actual curve is rising, whereas the predicted curve is not. Then, because the actual update values achieved larger numbers of requests (“count”), it will be classified as unexpected traffic, and it will generate a missed update classification because the actual traffic is falling but the prediction curve is growing.

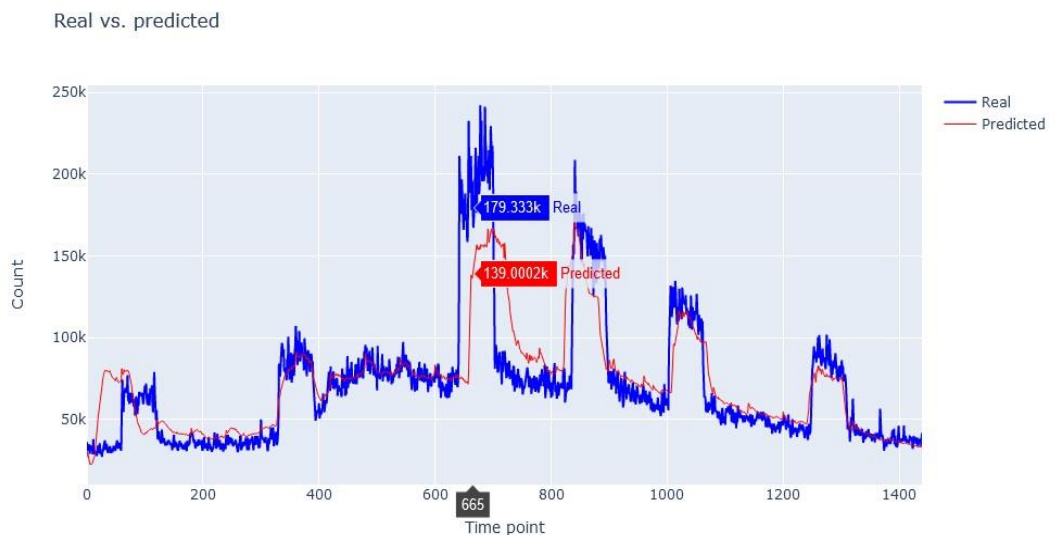


Figure 6. Real vs. one day ahead predicted traffic

We can alter these two variables when classifying in real-time and generating alerts. Either the classification window or threshold at which we decide whether to generate an alert. The classifier generates a large number of alerts when employing a window of 5 min, which provides a good real-time reaction time and a threshold of 20,000, as shown in Table 5. When we increase the threshold value to 30,000, we generate all the alerts that we can see with our eyes from Figure 6. Table 6 lists these alerts. Increasing the threshold further caused us to lose information about the first delayed update; therefore, we settled on a threshold of 30,000.

Table 5. Alerts generated for Figure 6

Time point (minutes)	Alert	Time point (minutes)	Alert
20-25	missed update	705-710	degraded performance
25-30	missed update	710-715	degraded performance
30-35	degraded performance	715-720	degraded performance
35-40	degraded performance	720-725	degraded performance
40-45	degraded performance	725-730	unexpected traffic
45-50	degraded performance	730-735	unexpected traffic
50-55	degraded performance	735-740	unexpected traffic
55-60	missed update	820-825	missed update
90-95	unusually large update	825-830	missed update
95-100	unusually large update	830-835	unexpected traffic
100-105	unusually large update	840-845	unusually large update
105-110	missed update	855-860	unusually large update
110-115	unusually large update	860-865	unusually large update
115-120	unusually large update	865-870	unexpected traffic
335-340	missed update	870-875	unusually large update
345-350	missed update	875-880	unusually large update
640-645	unusually large update	880-885	unexpected traffic
645-650	unusually large update	885-890	unusually large update
650-655	unusually large update	890-895	unusually large update
655-660	missed update	895-900	unusually large update
660-665	unexpected traffic	1000-1005	unexpected traffic
665-670	unusually large update	1005-1010	missed update
670-675	missed update	1010-1015	missed update
675-680	missed update	1050-1055	unusually large update
680-685	unusually large update	1065-1070	degraded performance
685-690	unusually large update	1070-1075	unusually large update
690-695	unusually large update	1275-1280	unusually large update
695-700	missed update	1280-1285	unusually large update
700-705	unusually large update		

Table 6. Alerts generated for Figure 6 using a threshold of 30,000

Time point (minutes)	Alert	Time point (minutes)	Alert
20-25	missed update	710-715	degraded performance
25-30	missed update	715-720	degraded performance
30-35	degraded performance	720-725	degraded performance
35-40	degraded performance	725-730	unexpected traffic
40-45	degraded performance	730-735	unexpected traffic
45-50	degraded performance	820-825	missed update
50-55	degraded performance	825-830	missed update
55-60	missed update	830-835	unexpected traffic
115-120	unusually large update	840-845	unusually large update
640-645	unusually large update	855-860	unusually large update
645-650	unusually large update	860-865	unusually large update
650-655	unusually large update	865-870	unexpected traffic
655-660	missed update	870-875	unusually large update
660-665	unexpected traffic	875-880	unusually large update
665-670	unusually large update	880-885	unexpected traffic
670-675	missed update	885-890	unusually large update
675-680	missed update	890-895	unusually large update
680-685	unusually large update	1000-1005	unexpected traffic
685-690	unusually large update	1005-1010	missed update
690-695	unusually large update	1010-1015	missed update
695-700	missed update	1065-1070	degraded performance
705-710	degraded performance		

6. RESULTS AND DISCUSSION

In the first stage, various regression models were tested to determine the best-performing model in predicting the network traffic curve. The selected model was then used as a component in the second stage, which focused on classifying anomalies in the network based on the predicted update curve. This two-stage assessment approach ensured that the developed system could accurately predict and classify network traffic anomalies.

6.1. Prediction evaluation

The first element of our proposed methodology is an LSTM neural network, which uses aggregated HTTP log data as input to solve a regression problem, in our case, predict the traffic load in the future so we

can compare it with real data curves in the future and classify occurring anomalies. It is very challenging to determine how accurate the model must be in order to assist us in finding the anomalies we seek. Because of this, we concentrated on comparing the actual curve to the predicted curve to determine how effectively the model can identify anomalies and whether it can classify them accurately. You can see a comparison of the model's prediction accuracy with other studies in the section 7 (discussion). Both the prediction and classification results are described in Table 5.

6.2. Expert evaluation

The second component of the system is a real-time windowing classifier that works with both the real traffic and the prediction curve of the regression model. We did not have a labelled dataset; therefore, we could not use this method to determine the correctness of the system. We relied on expert analysis against which the model alerts were matched. The following compares the ESET expert Ing. Matej Březina, who works in the Update Systems Department of ESET Internal Systems, and our proposed approach.

The expert stated that there were missed updates on February 24th, ranging from midnight to 10 am. As seen in Figure 7, the model was able to predict the updates, so we see two red peaks between time points 0 and 400, marked by the yellow rectangle. The next update occurred at 10 am, which corresponds to time point 600 in our sequence. According to the expert, the model anticipated an update at time point 1,060, which is too late, possibly because of the large volume of data without automatic update release (the updates are currently not released automatically, so the delay in updates is not an issue and the model is therefore not as accurate as it could be with automatic updates). The update took place at time point 1,000, as is typical. According to the expert, the rest of the sequence was normal as it did not cause any issues.

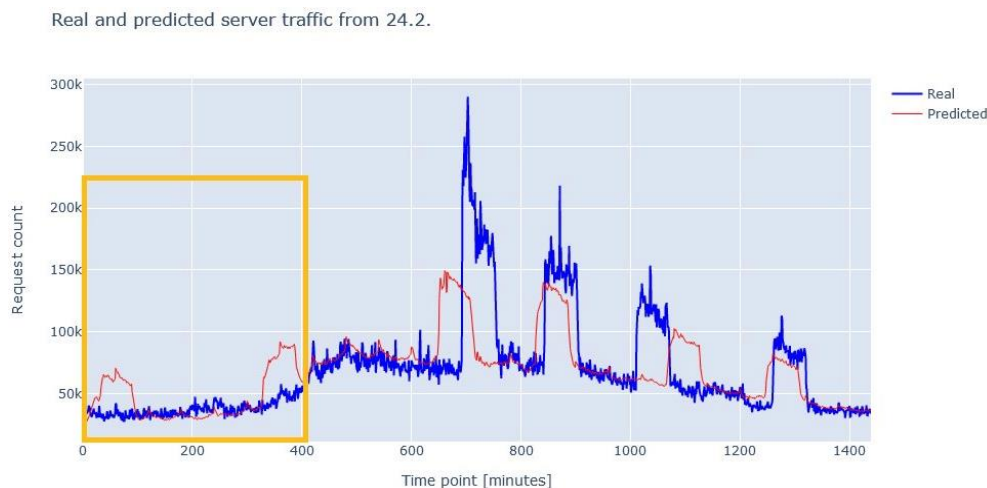


Figure 7. Real vs. one day ahead predicted traffic from February 24th

The classifier alerts, as shown in Table 7, correspond to the expert analysis and are thus considered correct. Because the classifier is designed to create alerts in real-time, it not only generates missed update alerts but also degraded performance updates when the curve is no longer rising or falling, indicating that the performance is degraded. The classifier again issued alerts at the time points referred to by the expert, as shown in Table 8. These categories correspond to the established threshold and trend guidelines. The classifier sent similar notifications for both missed updates: a missed update notice followed by a degraded performance alert, indicating that the update did not occur. The classifier detected more “missed updates” in both cases as the update curve began to rise again, but real traffic did not increase.

These alarms may be avoided by smoothing both the anticipated and actual update curves, such as with a rolling mean; however, this would reduce our accuracy, which is undesirable. It is preferable to send an alert as soon as a problem arises, and a smooth curve would cost us valuable time. The same is true for unexpected traffic, which occurs when the actual traffic increases, despite the prediction trend.

The server was flapping between 9 and 11 a.m. on February 26th, 2020, as shown in Figure 8, near the 600 time point in dark gray marked by the rectangle. Because certain server logs were missing due to external difficulties, the model input comprised four days of data to achieve a four-day-ahead prediction. The model was able to anticipate how typical traffic would look, which corresponds to the light gray line if the server was not flapping, as seen in red around the 600 time point.

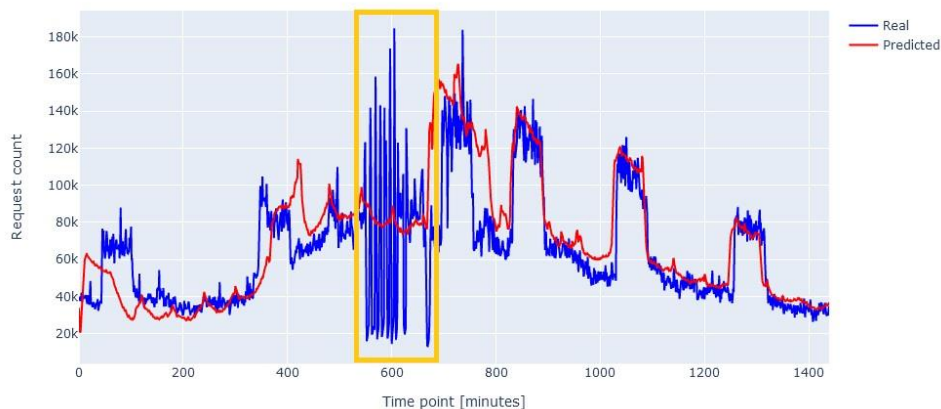
Table 7. Alerts generated for the first update expected after midnight

Time point (minutes)	Alert
35-40	missed update
40-45	degraded performance
45-50	degraded performance
50-55	degraded performance
55-60	missed update
60-65	degraded performance
65-70	unexpected traffic
70-75	degraded performance
75-80	degraded performance
80-85	degraded performance
85-90	degraded performance

Table 8. Alerts generated for the second update expected around 5 a.m

Time point (minutes)	Alert
330-335	missed update
335-340	degraded performance
340-345	degraded performance
345-350	missed update
350-355	degraded performance
355-360	degraded performance
360-365	degraded performance
365-370	degraded performance
370-375	degraded performance
375-380	unexpected traffic
380-385	degraded performance
385-390	degraded performance

Real and predicted server traffic from 26.2.

Figure 8. Real vs. 4 days ahead predicted traffic from February 26th

7. DISCUSSION

The first component of the system is an LSTM neural network that solves a regression problem with extremely particular data as input. As a result, there have not been many studies for comparison. The majority of the studies focused on network traffic prediction using the GEANT backbone network dataset, which includes traffic matrices that are different from our input data because they contain universal traffic data, and our update server data is not only very specific, but we only use features from HTTP logs. The prediction MSE for an LSTM network employing traffic matrices was approximately 0.042 [22], whereas our model attained an MSE of 0.0025, which is not comparable. Consequently, the following comparison is based on articles that deal with similar issues and use a similar, not identical, dataset, in our instance, network data.

Network packets between two virtual machines were collected using Wireshark containing HTTP, TCP, and ICMP requests from their local network to compare the performance of RNNs in this area, according to Ramakrishnan and Soni [23]. The results of comparing the planned system request count

prediction with their packet count forecasts are listed in Table 9. Both our models performed slightly better, although the results were similar.

Table 9. Comparison with Ramakrishnan and Soni [23]. We switched our proposed neural network using LSTM to GRU for this comparison

Model	GRU	LSTM
Comparison MSE	0.0030	0.0028
MSE of our proposed system	0.0028	0.0025

Different LSTM models for network traffic prediction have been compared by Nihale *et al.* [24]. RJ Hyndman compared six different time series in the paper. Data were captured every 5 min, hour, and day in the gathered time series. The daily findings were compared with the daily results from the publications, and the results are shown in Table 10. The better results of our proposed model can be explained by combining traffic from multiple servers in the same region, resulting in better forecasts as the curves are similar.

Table 10. Comparison with various models mentioned in the paper by Nihale *et al.* [24] using normalized root mean square error (NRMSE)

Model	Autoregressive integrated moving average (ARIMA)+RNN	Vanilla LSTM (VLSTM)	Delta LSTM (DLSTM)	Cluster LSTM (CLSTM)	Clusted Delta LSTM (CDLSTM)	Proposed system LSTM
NRMSE	0.115	0.120	0.114	0.112	0.109	0.042

The same dataset previously mentioned by RJ Hyndman was used in the study by Oliveira *et al.* [25] to compare multilayer perceptron using backpropagation (MLP-BP) as the training algorithm, multilayer perceptron with resilient backpropagation (MLP-RP), RNN, and deep learning stacked autoencoder (SAE). These models are compared with the proposed model in Table 11. These improved results can be explained by the unique use of our model. As we could not find any similar-sized NGINX logs dataset to compare with, we compared our model's performance on the web server access logs dataset containing 3.3 GB of web server logs from the Iranian shopping website zambil.ir. A sample log can be seen in Figure 9.

The shopping dataset did not contain all attributes we logged in our dataset and we, therefore, had to retrain our model so it only took the attributes present in the shopping dataset into account. This meant removing the body bytes sent attribute and also the server number as the target dataset only contained logs from one server and we used data gathered from 5 servers in the same region. After parsing and aggregating the shopping logs, the dataset consisted of 6,754 rows, whereas our dataset from one server consisted of 97 920 rows. We have extracted a random sample of the same length from our dataset and trained our model on both the ESET and shopping datasets. The retrained model achieved an MSE of 0.0225 on the ESET sample test set and an MSE of 0.035 on the shopping dataset test set.

The train performance of our model on the shopping dataset can be observed in Figure 10. By looking at the figure, we can see the model predicting the traffic curve very well and it is also trying to model the peaks and trying to smooth out some possible anomalies. For the model to be more precise, it needs more data for training. Still, this proves the usability of our model for predicting traffic from any web server log source.

Table 11. Comparison with various models mentioned in the paper by Oliveira *et al.* [25]

Model	MLP-BP	MLP-RP	RNN	SAE	Proposed system LSTM
NRMSE	0.200	0.202	0.197	0.336	0.042

```
31.56.96.51 - - [22/Jan/2019:03:56:16 +0330]
"GET /image/60844/productModel/200x200 HTTP/1.1"
200 5667 "https://www.zambil.ir/m/filter/b113"
"Mozilla/5.0 (Linux; Android 6.0; ALE-L21 Build/HuaweiALE-L21)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.158
Mobile Safari/537.36" "-"
```

Figure 9. Web server access logs Kaggle dataset sample

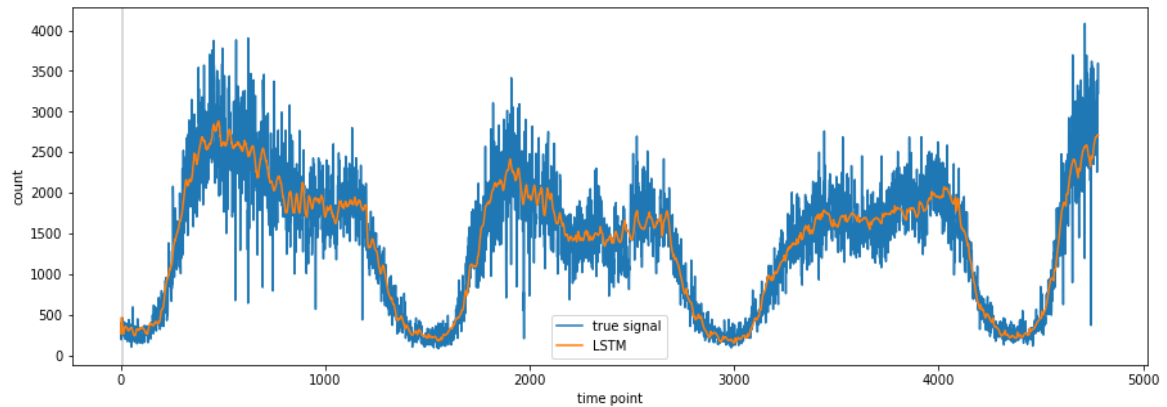


Figure 10. Model train performance on the web server access logs dataset

The test performance of our model on the shopping dataset can be seen in Figure 11. As we can see, the model learned the traffic peak behavior but was not able to model the peaks clearly. We observed this exact behavior when we trained the model on our two-week data sample and saw that the performance improved with more data. On three months of data, the model was able to predict the peaks, which made it possible to use the model for our anomaly detection. With the shopping data, we only had around 7% of the data we trained our model on. Unfortunately, all publicly available datasets are small and this was the largest we were able to find at the time of writing this article. As our model is trained on real traffic and not just filtered normal traffic without anomalies, we need millions of requests to be able to lower the importance of anomalies in the training set so we can later detect them in our test set. We still think that this proves the usefulness of our approach, it just needs a longer period of data to be able to model the peaks more precisely.

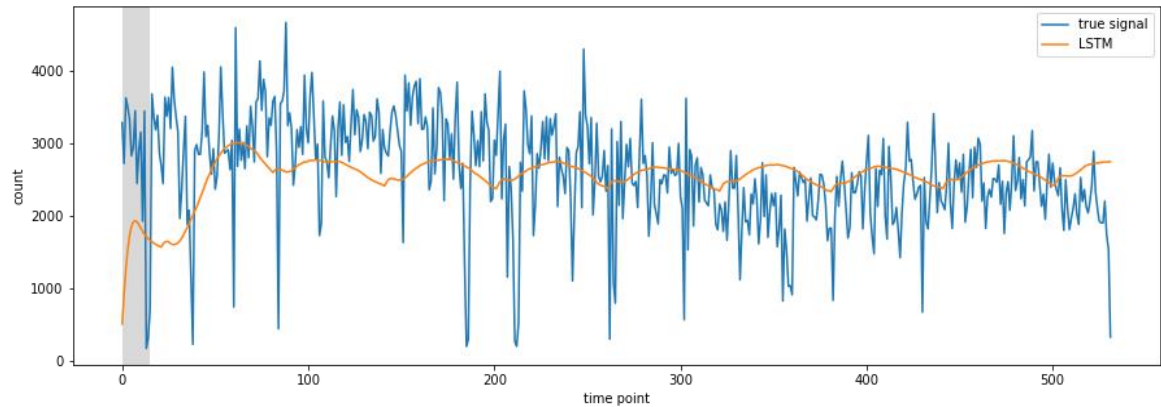


Figure 11. Model test performance on the web server access logs dataset

8. CONCLUSION

We proposed a method for detecting and classifying anomalies in HTTP logs that do not require extensive expert analysis for removing anomalies from real-time data or data labelling, which requires expert domain and system knowledge. Numerous studies claim to have achieved successful prediction and classification outcomes, but they frequently only did so on a single dataset and are not universally applicable to other datasets. We present a methodology applicable to any dataset consisting of HTTP logs. We examined the model in an actual network anomaly setting where we presented the value of our work. We were able to predict and categorize anomalies without the aid of a labelled dataset, demonstrating the model's usability while also gathering a substantial dataset from noisy network traffic. The proposed LSTM-based intrusion detection system was tested in a real-world setting and was successful in detecting various intrusions while in use, thereby demonstrating its usefulness. We divided anomalies into four categories using 5-minute windowing over both the predicted and real update curves: unexpected traffic (the load was high and we did

not expect an update), unusually large update (we issued an update and the load was higher than usual), missed update (the load was low but we did expect an update), and degraded performance (the load of an issued update was lower than usual). The system's overall findings and neural network performance, achieving an MSE of 0.0021 on our collected noisy dataset, were compared to existing solutions and expert analysis, where we achieved NRMSE and MSE values of an order of magnitude better. We hereby proved its usability as the system is able to provide sufficient update curve prediction accuracy on noisy web server traffic even without automatic updates and almost real-time reactions to occurring events, which is essential to take action against threats attacking the system and its infrastructure. The features used for anomaly detection in the paper were not novel but were chosen specifically because they represented a straightforward and generalizable methodology that could be applied to any HTTP log data. The goal was to create a model that could be used with minimal preprocessing or feature engineering, making it easy to apply to a wide range of different datasets and scenarios. By using well-established features that are commonly found in HTTP log data, we sought to create a model that would be simple to understand and implement, while still providing performance in terms of anomaly detection. Overall, the focus was on creating a straightforward and effective approach to anomaly detection, rather than on developing novel or cutting-edge features. The novelty of this work lies in using only HTTP web server logs to detect web server network intrusions in near real-time, without the need for a labelled or cleaned dataset. We not only compared our model's performance to that of other models trained on similar data, but we also deployed the model in a real-world setting of an antivirus company to demonstrate its practical value, as well as a study of how well the model performs in a real-world setting. The antivirus company validated the value of our approach for their ability to react promptly to emerging anomalies. The techniques we described are not novel, but we presented a new methodology to apply existing techniques to live data that already has anomalies without having to label them, allowing us to identify anomalies and assist professionals in their work. This methodology enables the use of any dataset composed of HTTP logs as input data, providing a robust solution that performs effectively on a variety of datasets.

ACKNOWLEDGEMENTS

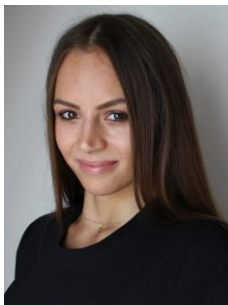
We would like to thank ESET Research Centre for its cooperation. We would also like to express our gratitude to Matej Březina for providing extensive information regarding ESET update servers. This publication has been written thanks to the support of the Operational Programme Integrated Infrastructure for the project: Research in the SANET network and possibilities of its further use and development (ITMS code: 313011W988), co-funded by the European Regional Development Fund (ERDF).




REFERENCES

- [1] X. Ni, D. He, S. Chan, and F. Ahmad, "Network anomaly detection using unsupervised feature selection and density peak clustering," in *Applied Cryptography and Network Security*, Springer International Publishing, 2016, pp. 212–227.
- [2] I. Syarif, A. Prugel-Bennett, and G. Wills, "Unsupervised clustering approach for network anomaly detection," in *Networked Digital Technologies*, Springer Berlin Heidelberg, 2012, pp. 135–145.
- [3] N. Chouhan, A. Khan, and H.-R. Khan, "Network anomaly detection using channel boosted and residual learning based deep convolutional neural network," *Applied Soft Computing*, vol. 83, Oct. 2019, doi: 10.1016/j.asoc.2019.105612.
- [4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *Network traffic anomaly detection and prevention: concepts, techniques, and tools*. Springer, 2017.
- [5] S. Mascaro, A. E. Nicholso, and K. B. Korb, "Anomaly detection in vessel tracks using Bayesian networks," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 84–98, Jan. 2014, doi: 10.1016/j.ijar.2013.03.012.
- [6] A. A. H. Riyaz, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A Survey," *International Journal of Information Management*, vol. 45, pp. 289–307, Apr. 2019, doi: 10.1016/j.ijinfomgt.2018.08.006.
- [7] L. Nie, D. Jiang, and Z. Lv, "Modeling network traffic for traffic matrix estimation and anomaly detection based on Bayesian network in cloud computing networks," *Annals of Telecommunications*, vol. 72, no. 5–6, pp. 297–305, Jun. 2017, doi: 10.1007/s12243-016-0546-3.
- [8] S. Pauwels and T. Calders, "Extending dynamic bayesian networks for anomaly detection in complex logs," *arXiv preprint arXiv:1805.07107*, May 2018.
- [9] G. Poojitha, K. N. Kumar, and P. J. Reddy, "Intrusion detection using artificial neural network," in *2010 Second International conference on Computing, Communication and Networking Technologies*, Jul. 2010, pp. 1–7, doi: 10.1109/ICCCNT.2010.5592568.
- [10] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, Jan. 2016, doi: 10.1016/j.jnca.2015.11.016.
- [11] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, Dec. 2014, pp. 4–11, doi: 10.1145/2689746.2689747.
- [12] V. L. Cao, M. Nicolau, and J. McDermott, "A hybrid autoencoder and density estimation model for anomaly detection," in *Parallel Problem Solving from Nature PPSN XIV*, Springer International Publishing, 2016, pp. 717–726.
- [13] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, Mar. 2017, doi: 10.1049/iet-its.2016.0208.




- [14] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Nov. 2016, pp. 324–328, doi: 10.1109/YAC.2016.7804912.
- [15] A. Lazaris and V. K. Prasanna, "An LSTM framework for modeling network traffic," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 19–24.
- [16] H. Lu and F. Yang, "A network traffic prediction model based on wavelet transformation and LSTM network," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Nov. 2018, pp. 1–4, doi: 10.1109/ICSESS.2018.8663884.
- [17] H. D. Trinh, L. Giupponi, and P. Dini, "Mobile traffic prediction from raw data using LSTM networks," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2018, pp. 1827–1832, doi: 10.1109/PIMRC.2018.8581000.
- [18] Q. Zhuo, Q. Li, H. Yan, and Y. Qi, "Long short-term memory neural network for network traffic prediction," in *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Nov. 2017, pp. 1–6, doi: 10.1109/ISKE.2017.8258815.
- [19] S. Naseer *et al.*, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48231–48246, 2018, doi: 10.1109/ACCESS.2018.2863036.
- [20] R. K. Malaiya, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim, "An empirical evaluation of deep learning for network anomaly detection," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, Mar. 2018, pp. 893–898, doi: 10.1109/ICNC.2018.8390278.
- [21] T.-Y. Kim and S.-B. Cho, "Web traffic anomaly detection using C-LSTM neural networks," *Expert Systems with Applications*, vol. 106, pp. 66–76, Sep. 2018, doi: 10.1016/j.eswa.2018.04.004.
- [22] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2017, pp. 2353–2358, doi: 10.1109/ICACCI.2017.8126198.
- [23] N. Ramakrishnan and T. Soni, "Network traffic prediction using recurrent neural networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018, pp. 187–193, doi: 10.1109/ICMLA.2018.00035.
- [24] S. Nihale, S. Sharma, L. Parashar, and U. Singh, "Network traffic prediction using long short-term memory," in *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Jul. 2020, pp. 338–343, doi: 10.1109/ICESC48915.2020.9156045.
- [25] T. P. Oliveira, J. S. Barbar, and A. S. Soares, "Computer network traffic prediction: a comparison between traditional and deep learning neural networks," *International Journal of Big Data Intelligence*, vol. 3, no. 1, 2016, doi: 10.1504/IJBDI.2016.073903.

BIOGRAPHIES OF AUTHORS



Lenka Benova    received her Bachelor's degree in Computer Engineering in 2019 from the Faculty of Informatics and Information Technologies of the Slovak University of Technology in Bratislava. In 2021, she received her Master's degree at the before mentioned university in the field of Computer Science. She is currently a Ph.D. student of Applied Informatics at the Faculty of Informatics and Information Technologies, Slovak University of Technology with a focus on increasing web server security. She can be contacted at email: lenka.benova@stuba.sk.



Ladislav Hudec    received his Ing diploma summa cum laude in electronics from the Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University, Prague, in 1974. In 1985, he received a C.Sc degree (Ph.D.) in Computer Machinery from the Faculty of Electrical Engineering, Slovak Technical University, Bratislava, in 1989 he was appointed Associate Professor. Since 1974 he is with the Slovak Technical University. During the period 1992-1993, he served as Director of the SARC-Centre for Advancement, Science and Technology, Bratislava. During the period from 1993-2010, he served as National Coordinator at the European Cooperation in Science and Technology (COST). He is currently an Associate Professor of Computer Science and Engineering, Deputy Director of the Institute of Computer Engineering and Applied Informatics, Faculty of Informatics and Information Technology, Slovak University of Technology. He reads lectures on Principles of information technologies security, Security of information technologies and Internet security. He is the author or co-author of over 60 scientific papers published in journals and proceedings of conferences and over 80 technical papers in the field of fault-tolerant computing, embedded systems, and computer security. He led over 90 research grants and industrial contracts. He is a member of the Information System Audit and Control Association (ISACA), and holder of the CISA license. He can be contacted at email: ladislav.hudec@stuba.sk.