Using Web Server Logs to Identify and Comprehend Anomalous User Activity

1st Lenka Benova
Institute of Computer Engineering and Applied Informatics
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
lenka.benova@stuba.sk

2nd Ladislav Hudec
Institute of Computer Engineering and Applied Informatics
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
ladislav.hudec@stuba.sk

2023 17th International Conference on Telecommunications (ConTEL) | 979-8-3503-0223-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CONTEL58387.2023.10199092

***Abstract*—This research paper presents a study for identifying user anomalies in large datasets of web server requests. Using a cybersecurity company's network of web servers as a case study, we propose a technique for analyzing user activity in NGINX logs. The proposed method does not require a labeled dataset and is capable of efficiently identifying different user anomalies in large datasets with millions of daily requests. The results of the analysis provided a deeper understanding of user behavior when seeking updates through web requests and aided in interpreting the findings. Clustering the anomalies helped to produce typical clusters and further supported the interpretation of the results. This work provides valuable insights into user behavior in web server networks and highlights the importance of efficient anomaly detection techniques in large datasets. The findings have potential real-world applications in the field of cybersecurity, particularly in providing network security analysts with an automated and more objective approach to threat analysis. This study showcases the importance of automated methods for analyzing user activity in web server networks and provides a more objective and efficient approach to detecting user anomalies in large datasets. This approach contributes to the development of more effective and precise cybersecurity systems, ultimately improving the protection of network infrastructures from malicious attacks.**

***Index Terms*—anomaly detection, web server, isolation forest, network logs, user behaviour, clustering, DBSCAN**

I. Introduction

This paper is an extension of work originally presented in 2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC) [1] and it is being expanded by further extension of the proposed method, its findings and comprehension of the found outliers using clustering and case studies.

Due to advancements in Internet technology and the rise in network threats, network intrusion detection has become an important research topic [2]. Recent years have seen a lot of interest in machine learning, and new, more effective machine learning algorithms are appearing. Machine learning techniques are used in numerous networking and computer applications, and they demonstrate enhanced performance.

One of the most critical components of network security is intrusion detection. Finding unusual behaviour in a network and computer system brought on by attackers is the primary objective of intrusion detection. It is challenging to design and put into practice an intrusion detection system that aims to detect unidentified anomalies [3].

The majority of machine learning-based anomaly detection methods concentrate on total network traffic. However, it is necessary to look at the behaviour of particular web server users and the network traffic curve when evaluating web server anomalies. Individual anomalies are frequently overlooked because they get lost in the daily stream of logs. As a result, it's critical to be alert to anomalous activity and take appropriate action when it's noticed. It offers it either by limiting the user's access to the server or discovering issues with the web server's internal configuration or the data.

The authors would like to thank for financial contribution from the STU Grant scheme for Support of Young Researchers.

II. Related work

Numerous methods for recognizing anomalies by analyzing user behaviour have recently been developed. Most are tested on labelled datasets with a clustering or pattern-matching focus, while others employ advanced machine learning techniques to adaptively identify unusual patterns in real-time.

Web logs from the BUPT school websites, where NGINX was being used as a reverse proxy, were gathered by the authors Gao, Ma and Li [4]. 4877 user sequences were present in the dataset, which was compiled from access logs collected over five days in 2017 from March 16 to March 20. They retrieved 52 713 attack logs from the MACCDC2012 dataset and analyzed them to determine the detection and false alarm rates. Based on the obtained attributes, the authors looked at user behaviours and data from web server logs. The authors extracted four crucial characteristics from the log and developed a user access sequence to simulate user behaviour. The authors used two algorithms to identify network threats based on user behaviour. One was a simple technique for calculating the entropy of a user sequence, and the other used machine learning to cluster different users' behaviour. The authors used entropy as the eigenvalue of the k-means model to calculate the distance to the cluster centre for different users. The detection rate was 100% and the false alarm rate was 1.68% when the distance was between 1.5 and 1.6, indicating that all 12 fraudulent users had been found. With a false alarm rate of 1.33%, the detection rate for distances between 1.7 and

TABLE I
METRICS OF STUDIED APPROACHES ON EXPERIMENTAL DATASETS [6].

Approach	PCA			LogCluster			IM			DeepLog			LogAnomaly		
Dataset	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
D1	0.40	1.00	0.57	0.38	1.00	0.55	0.51	1.00	**0.68**	0.64	0.33	0.44	0.45	0.67	0.54
D2	0.56	0.43	0.48	0.50	0.57	0.53	0.34	0.43	0.38	0.71	0.86	**0.78**	0.71	0.86	0.78
D3	0.67	0.50	0.57	0.72	0.50	0.59	0.43	1.00	**0.60**	0.43	0.50	0.46	0.62	0.50	0.55
D4	0.20	0.44	0.28	0.24	0.53	**0.33**	0.32	0.21	0.26	0.17	0.38	0.23	0.22	0.34	0.27
D5	0.92	1.00	**0.96**	0.80	1.00	0.89	0.87	1.00	0.93	0.91	1.00	0.95	0.91	1.00	0.95
D6	0.64	1.00	0.78	0.86	0.81	0.83	0.82	1.00	0.90	0.93	0.96	0.94	0.92	1.00	**0.96**
D7	0.42	0.50	0.46	0.44	0.50	0.47	0.58	0.25	0.35	0.57	1.00	**0.73**	0.53	1.00	
D8	0.10	0.25	0.14	0.35	0.25	0.29	0.14	0.25	0.18	1.00	0.50	**0.66**	1.00	0.50	**0.66**
D9	0.18	0.60	0.28	0.28	1.00	0.44	0.44	0.40	0.42	0.32	1.00	0.48	0.54	0.60	**0.57**
D10	0.29	0.33	0.31	0.43	0.33	0.37	0.29	1.00	0.45	0.74	0.50	0.60	1.00	0.50	**0.67**
Mean F1 (std)	0.48 (0.24)			0.53 (0.19)			0.52 (0.24)			0.63 (0.22)			0.66 (0.19)		

2.0 was 96.67%, meaning that 11 out of 12 fraudulent users were found.

Debnath et al. [5] proposed LogLens, a real-time log analysis system that detects anomalies with little to no human intervention. The system divides anomaly detection methods into two categories: stateless and stateful. Spoofing attacks were identified by the log sequence anomaly detector of LogLens in a set of Signaling System No. 7 (SS7) logs.

Zhao et al. [6] studied the effectiveness of five widely used unsupervised log anomaly detection algorithms to answer two research questions:

1) How well do these approaches perform on different real-world log data containing various anomalous patterns?
2) What is the effectiveness of these techniques?

The studied algorithms include statistical modeling (PCA [7], LogCluster [8], and IM [9]) and deep learning-based approaches (DeepLog [10] and LogAnomaly [11]). Ten datasets were prepared, consisting of various log types, such as application errors, user operations, and system logs. The effectiveness of these approaches was compared based on precision, recall, and micro F1-score metrics.

The results showed that deep learning-based methods outperformed conventional statistical methods in terms of detecting anomalies in real-world data. However, the studied methods did not perform consistently well on all datasets. The average micro F1-score across all datasets was only 0.48066 as seen in Table I.

Based on the findings, the authors proposed LogAD, a general log anomaly detection system, to deal with these issues. LogAD is based on the integration of a knowledge base based on domain experience, the visual presentation of an interpretable report, and the combination of various anomaly detection algorithms to handle a wide range of abnormal patterns. The authors combined many methods for anomaly detection and applied the ensemble learning principle in their system. Engineers can then select one or more suitable anomaly detection components using their domain knowledge to find particular anomalous patterns in different logs. While earlier work employs a variety of methods (including PCA, LogCluster, and IM) to more accurately identify anomalies, it is not designed to handle a wide variety of anomalous patterns. The authors suggested a multivariate time-series anomaly detection method for template count based on LSTM. They were looking for anomalies utilising the commonly utilised distribution distance measurement (JS divergence). The template sequence and variable value techniques were developed based on earlier research. The average F1-score of 0.83 for LogAD confirms its effectiveness.

With the help of the clustering module, LogMine [12] swiftly clusters a huge batch of input logs under severe restrictions. Author's pattern recognition module then creates a pattern for each cluster. The leaf level of the pattern hierarchy is made up of the sets of patterns. In later iterations, the algorithm continues to produce clusters with less restrictions and merge the clusters to create more generic patterns, which will represent a new parent level in the hierarchy. Until the most general pattern is attained or the hierarchy of patterns is fully constructed, LogMine iterates continuously. Massive heterogeneous logs can include patterns that LogMine can find. It is the first framework of its kind that is unsupervised, scalable, and heterogeneity-resistant. On a distributed platform, LogMine is capable of processing millions of logs in a couple of seconds. With its one-pass architecture and small memory footprint, the framework can scale to handle hundreds of millions of logs.

III. DATASET

The dataset used in this study was obtained from the application servers of the ESET, spol. s r.o., a well-known cybersecurity company based in Slovakia. The company specializes in firewall and antivirus software and its client software automatically downloads virus signatures and program module updates from the company's update servers within a defined time period.

The Zabbix monitoring system is used to monitor these updates and the NGINX logs client connections requesting updates. This logged data is used as a starting point to identify any potential anomalies, such as targeted attacks or questionable behavior. It is important to note that the number and nature of anomalies present in the dataset is unknown. The objective of this study was to investigate the dataset and find threshold values for the applied attributes.

The dataset consists of NGINX web server logs from several ESET update servers located all over the world. The logs include specific details regarding the connection between the client and the server, which distributes the update. Once a client request is processed, NGINX records the information about it in the access log. Table II contains a comprehensive list of the attributes that are currently logged. The dataset contained 15GB of compressed logs, making it a substantial size for data analysis.

IV. FEATURE EXTRACTION

This initial dataset was supplied into a parser, which used a user list identifying users by their hardware fingerprint and the times of day they were requesting an update as input. Due to GDPR, IP addresses could not be used (EU General Data Protection Regulation). The http_user_agent field was also used to parse the number of distinct HTTP requests, such as GET and HEAD, as well as the response codes, which included the client's system details and language. You can see the dataset in Table III, which maps the hardware fingerprint

TABLE II
NGINX LOGGING ATTRIBUTES.

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 milliseconds resolution
http_user_agent	identification of the client originating the request

of the user to particular web server access timings. The dataset includes a map that shows all of the requests that a user has made during a given period of time.

Each feature name denoted by a number in Table III corresponds to a 30-minute period. The number included in feature '3', for instance, counts the number of requests made in the 90th minute of the day since feature '3' indicates the third 30-minute interval of the day. As there are 24 hours in a day and 60 minutes in each hour, we receive 1 440 minutes in a day, which is divided into 48 intervals of 30 minutes each.

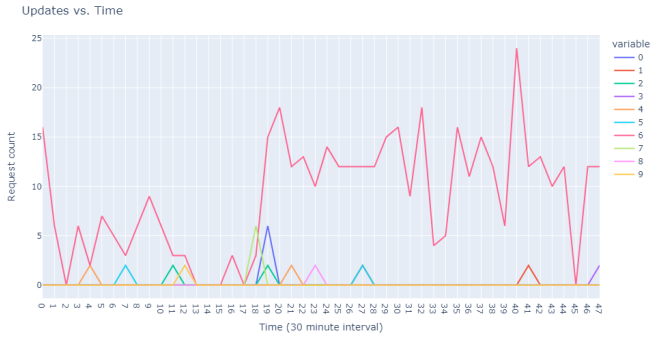

Fig. 1. User request time plot

The dataset is unlabeled for a reason—user-based anomalies haven't been investigated on this dataset before, therefore we don't know which user request sequences should be classified as abnormal. Therefore, we are employing an anomaly detection technique that can identify these anomalies without requiring a thorough examination and labelling by experts. The plotted user request behaviour in Figure 1 shows how varied the behaviour of just 10 randomly selected users is, and it would take a specialist a significant amount of time to study the most unusual behaviours within just a small portion of the dataset.

The extracted dataset includes 8 484 251 distinct users who performed queries on a particular web server in Europe over the course of one day.

V. Our Method

The techniques used in this study make use of Isolation Forests, which, in contrast to conventional techniques, do not rely on density or distance to discover anomalies, hence saving time due to the absence of time-consuming calculations. It has had remarkable success in anomaly identification and is perfectly suited for our large dataset which we obtained from web servers since, unlike clustering, which requires a lot of system memory, it can quickly find anomalies within datasets [13].

We first used DBSCAN to group users into clusters and then analyze the outliers to identify common user behaviour. However, this method was computationally inefficient because we could only cluster a portion of 100,000 users using 64GB of RAM, so we opted to use isolation forests instead, which do not attempt to model typical data and are therefore much more memory-efficient.

Two properties of anomalous data form the foundation of the Isolation Forest method. The amount of the data collection as a whole is lower than it is for the abnormal data, and the attribute value of the normal data is very different from that of the anomalous data. In a training set, the data are gradually separated using solely numerical values until iTree can distinguish one set of data from the other. Due to their great sensitivity to segregation, the anomalous data are located closer to the root node of the tree than the regular data, and can only be identified under a few specific situations [13].

We employed two different approaches to identify anomalies in the dataset [1]. The first approach aimed to identify anomalous user behavior by analyzing 8,484,251 records and 57 features, including counts of HTTP methods, status codes, and update requests made in 30-minute day intervals. We used an isolation forest model with default parameters, which identified 60,000 anomalies. We performed descriptive statistics to summarize the central tendency, dispersion, and shape of the dataset's distribution IV.

The second approach focused on identifying anomalous user behavior within request sequences. We modified the dataset to consider only the ratio of requests within a certain interval to the total number of requests a user made, and used an isolation forest model with 48 features representing user request time and 100 estimators. The model detected 18,000 anomalous sequences within one day of data, using a contamination of 0.05 to mark the top 0.5% of the data as anomalous. Our approach sorted these sequences by the model's assigned anomaly score, which can be found in Table VI. As we can see in Figures 2 and 3, the two models identified different anomalies.

VI. Anomalous Data Interpretation

The first 1,000 most unusual sequences discovered by the isolation forest were taken and plotted using a line plot. We were attempting to spot trends in abnormal update request sequences.

Already from the first graph in Figure 4, we can observe distinct update download sequences, representing the behavior of multiple users, that differ from the others. If we zoom in on the graph as seen in Figure 5, we can see the second extreme more closely. If we zoom in on the graph even more as seen

TABLE III
DATASET SAMPLE (HARDWARE FINGERPRINTS WERE MASKED DUE TO PRIVACY).

hardware_fingerprint	get_count	head_count	country	system	status_200	...	status_304	0	...	47
XXXXXX...0	3	3	en_us	Windows	3	...	0	0	...	0.00
XXXXXX...1	2	2	es_es	Windows	0	...	2	0	...	0.00
XXXXXX...2	1	1	pl_pl	Windows	1	...	0	0	...	2
XXXXXX...3	5	425	en_us	Windows	10	...	420	0	...	12

TABLE IV
DESCRIPTIVE STATISTICS FOR THRESHOLD IDENTIFICATION.

	get_count	head_count	status_200	status_401	status_304	status_404	status_416	status_206	0	...	47
mean	108.69	137.17	36.93	114.10	94.62	0.07	0.00	0.14	3.66	...	2.82
std	977.59	766.55	544.45	950.12	580.02	3.09	0.00	3.98	25.67	...	28.78
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00
25%	6.00	29.00	4.00	1.00	7.00	0.00	0.00	0.00	0.00	...	0.00
50%	25.00	60.00	9.00	18.00	38.00	0.00	0.00	0.00	0.00	...	0.00
75%	104.00	143.00	21.00	110.00	92.00	0.00	0.00	0.00	2.00	...	2.00
max	160 662.00	163 096	90 439.00	183 238.00	100 365.00	384.00	1.00	608.00	3 106.00	...	4 301.00

TABLE V
ANOMALOUS USERS SAMPLE.

	get_count	head_count	country	system	status_200	status_401	status_304	status_404	status_416	status_206	0	...	47
user0	160 662	163 096	he_il	Windows	90 439	183 238	49 847	55	0	175	3 106	...	3 492
user1	63 246	0	tr_tr	Android	63 245	0	0	0	0	0	169	...	0
user2	100 377	0	tr_tr	Android	12	0	100 365	0	0	0	0	...	50
user3	60 437	0	pl_pl	Android	60 437	0	0	0	0	0	0	...	0

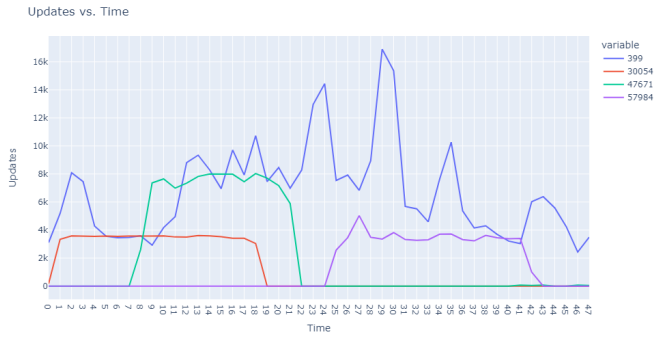

Fig. 2. Anomalous user's request behaviour.

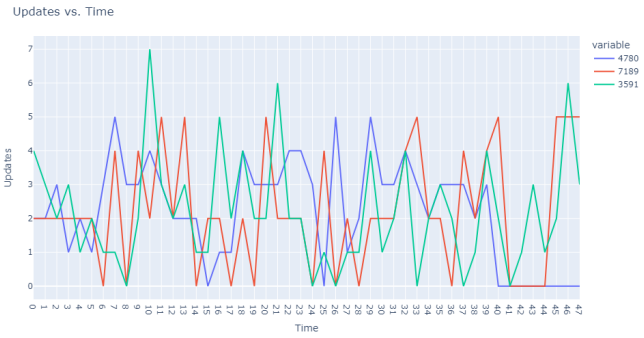

Fig. 3. Anomalous user request sequences.

TABLE VI
UPDATE BEHAVIOUR DATASET SAMPLE.

	0	1	2	3	...	47
user1	2%	2%	3%	1%	...	0%
user2	2%	2%	2%	2%	...	5%
user3	4%	3%	2%	3%	...	3%
user4	1%	3%	2%	1%	...	1%
user5	3%	3%	4%	1%	...	1%

in Figure 6, we can clearly see the behaviour of the other sequences marked as anomalies. However, the question is how to interpret such a number of anomalies.

VII. CLUSTERING

In order to examine large datasets and find patterns or groups within the data, clustering is a powerful strategy.

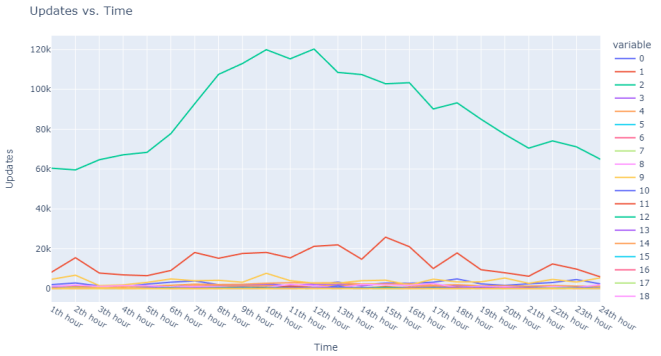

Fig. 4. Overview of user request behavior.

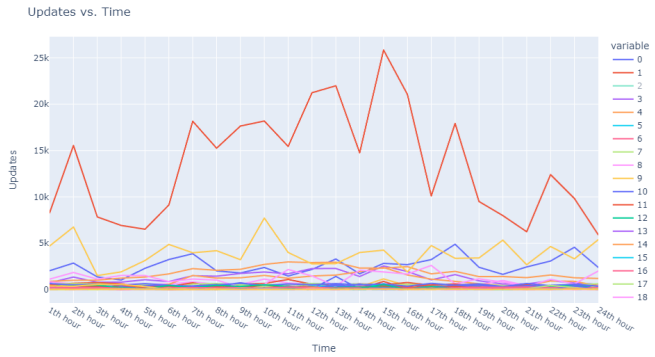

Fig. 5. Focused user request behavior: zoomed-in view.

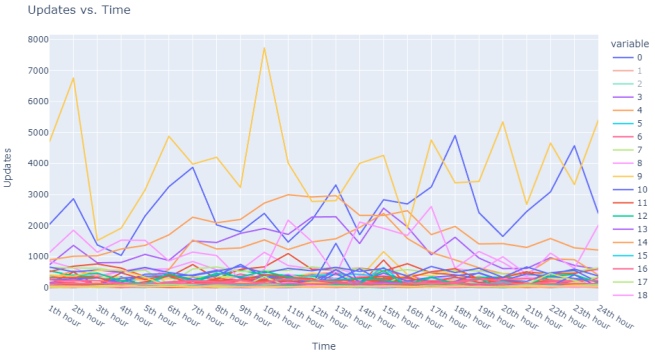

Fig. 6. Detailed user request analysis: zoomed-in view.

Clustering can assist in identifying and isolating anomalous data points when used to large anomaly outcomes from an isolation forest, enabling a more concentrated and effective investigation. This can be very helpful in finding and treating outliers, which are frequently concealed in large datasets.

In addition, clustering techniques can offer insightful information about the connections between data points, enabling the discovery of potential correlations or dependencies. This knowledge can be applied to further analyze the data and comprehend its underlying structure. Clustering can be used to reduce the number of dimensions in the data, which improves the visualisation and comprehension of complex patterns. This can assist to simplify the data and make it more manageable for analysis, which can be especially helpful when working with huge datasets with several attributes.

A. Data point selection

Clustering performance depends heavily on how the data points are chosen. We first used the Isolation Forest method mentioned earlier to find the top 1,000 anomalies in order to get a representative sample of the anomalous data points. Isolation Forest results were chosen because of its effectiveness in locating outliers in high-dimensional data, which frequently characterizes web server request logs. By choosing the top 1,000 anomalies, we made sure that the clustering algorithm's input contained data points that displayed unusual patterns, which consequently made it less challenging to identify recognizable anomalous clusters. This strategy allows us to design suitable countermeasures for preserving the stability and security of the web server and better comprehend the underlying structure of the anomalous user requests.

B. Scaling

To avoid variables with a higher scale dominating the similarity measure, which is employed in clustering methods, data must be scaled before clustering. Scaling equalizes the magnitude of each variable and reduces bias against those with a bigger scale. Additionally, it enables an equal comparison of variables with various units of measurement and prevents features with large volatility from dominating the clustering outcome. Results from clustering may be more accurate and informative when the data are scaled. We decided to scale our data using StandardScaler, which sets the data's mean at 0 and standard deviation at 1. StandardScaler is frequently used for scaling data in machine learning due to its simplicity and efficiency.

C. Clustering model

Based on the results of our analysis and the assumption that the clusters will have a similar density, we chose the DBSCAN model for clustering the anomalies found using the Isolation forest model. DBSCAN is used to manage clusters of various shapes and sizes and is not significantly impacted by noise or outliers, which makes it ideally suited for our noisy dataset from real traffic.

D. DBSCAN parameters

DBSCAN [14] bases its behavior on two primary parameters. The number and size of the clusters discovered by DBSCAN depend on the values used for eps and min_pts. More samples will be included in the same cluster when the eps value is high, whereas fewer and smaller clusters will form when the eps value is low. A large min_pts number will produce fewer, larger clusters, whereas a small min_pts value will produce more, smaller clusters. To get useful results from DBSCAN, it's critical to select the right values for eps and min_pts.

In order to discover clusters in the data, min_pts is often set to the value of feature count multiplied by two. This value is based on the presumption that the data contains clusters with comparable densities and that the ratio of the number of samples needed to produce a dense region to the number of features in the data is proportional.

For example, if data contains n features, min_pts is set to 2n. This means that a sample needs at least 2n other samples within eps of it in order to form a dense zone. This makes it more likely that the discovered clusters will be dense and have significant structures. In our case, min_pts was also set to feature count times two.

In our case, we used the elbow method, meaning we used a min_pts value of 96 (48 features x 2) and the default eps value, as no fine-tuning was needed to provide us with valuable results.

E. Visualisation

The dataset in the Figure 7 was plotted as a scatter plot, with clusters denoted by a different colour. The X-axis represents the get_count and the Y-axis the head_count.

We can see right away from the graph that the users who made the most GET and HEAD queries are grouped together. It's also interesting how smaller GET and HEAD values nearly exclusively generate all other clusters. We needed more in-depth research to determine why this is the case and why there are more clusters in this range.

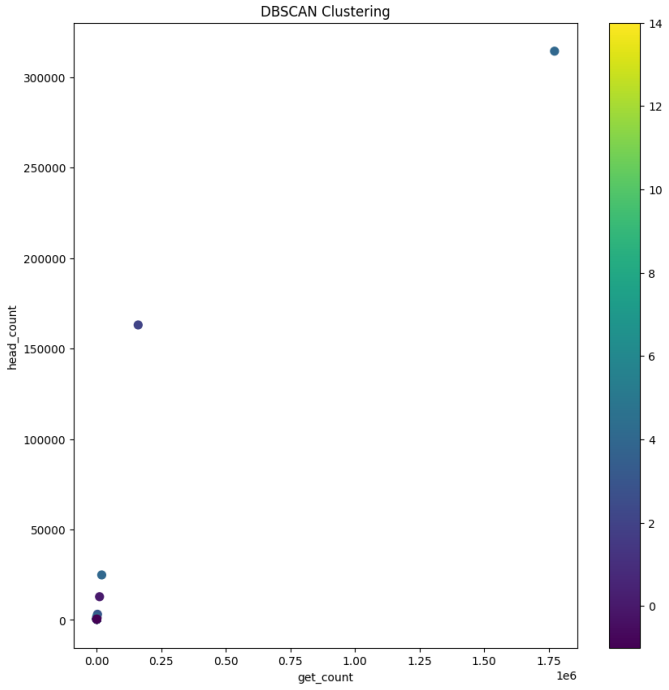

Fig. 7. DBSCAN clustering: GET and HEAD count.

F. Detailed analysis

For a detailed analysis, we had to reduce the dimensionality. For this purpose, we used t-SNE [15], which is unsupervised, non-linear t-Distributed Stochastic Neighbor Embedding. It is a method for exploring and visualizing high-dimensional data. Local similarities and spatial correlations between data points are kept in their original, higher dimensionality by t-SNE while producing a 2-D visual representation of multi-dimensional data. In essence, it is used to aid in the comprehension of the distribution and arrangement of data in high-dimensional space.

We used t-SNE and Plotly to examine and visualize the results after running DBSCAN. We used the same data array we supplied into the DBSCAN model to run t-SNE after our DBSCAN model had finished running and we had appended the cluster labels to our original dataset. Then, we added the t-SNE-identified x and y components to our initial dataset.

The resulting plot can be observed in Figure 8, where each colour represents a different cluster.

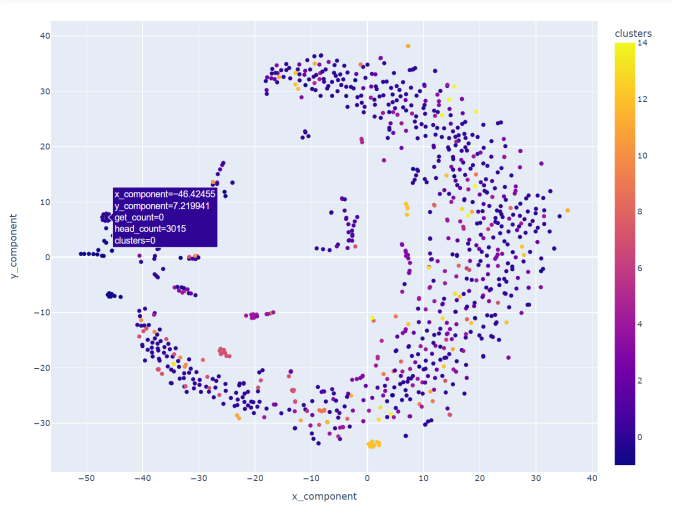

Fig. 8. DBSCAN clustering: after t-SNE.

From this figure, we were able to deduce that users who generated the same or slightly different amounts of GET and HEAD queries constituted a cluster. This can be observed in Figure 9 where data points form a linear function.

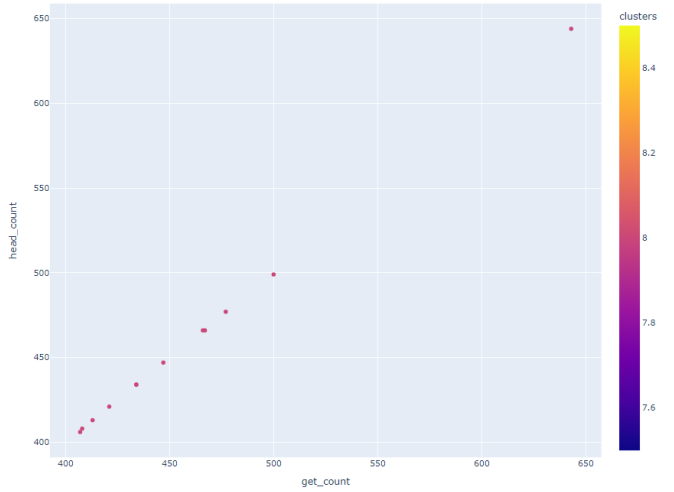

Fig. 9. Cluster with similar GET and HEAD counts.

Then, there were multiple clusters clustering users according to their countries like Spain, Russia, Poland, France, and Italy. We found out that users from Slovakia, the Czech Republic, Germany and Hungary make similar or the same amount of GET and HEAD requests while the other countries don't.

When compared to the hundreds of thousands of HEAD requests, people who generated very few GET requests formed a cluster. These users originated from across the globe. However, one sample from this cluster generated 1.77 million GET requests and roughly 314 000 HEAD requests despite not having a nation listed. The samples with the greatest disparity in GET and HEAD counts appear to have formed this cluster.

G. Clustering user request patterns

We also clustered user requests according to their request times throughout the day. We omitted all the other attributes so that the model is not influenced by them. We also were not working with the exact number of requests a user has made in a specific time frame (in our case each 30 minutes of the day) but we rather looked at the percentage of his requests in that specific time-frame. The reason for this decision is we did not want the model to be influenced by the high request values, we already observed those with isolation forest and our first clustering approach. We were trying to find anomalous request patterns of multiple users, which formed a cluster. This way, we can observe anomalous patterns in our network and implement countermeasures so that our security experts are already aware of the anomaly ones it starts happening and can be alerted to take further action.

We first used the elbow method to find anomalous clusters in our data. This meant using a min_pts value of 2 x our features, which resulted in the number 96. This method was not suitable for our dataset as it only resulted in one cluster and some noise points, which did not meet our requirements. We aimed for finding typical anomalous activity more users had in common, not one cluster with all anomalies and noise points with even more anomalies. After running DBSCAN multiple times with varied min pts and eps values on user update request times (features 0-47), we typically obtained results with one or two clusters. The first one contained only noisy samples and the second one carried the remaining data. When using 3 samples for min_pts, we observed 5 users in cluster 0 as seen in Figure 10, other were noise.

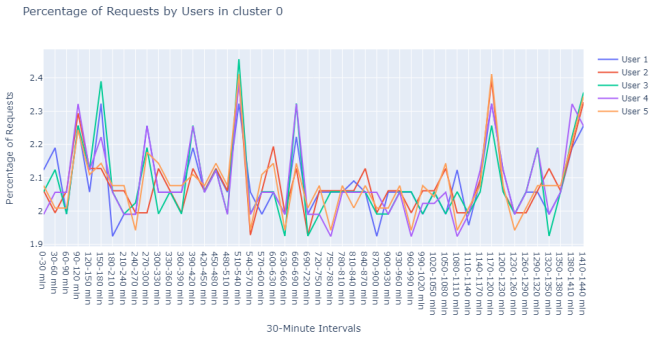

Fig. 10. DBSCAN with 3 min_pts for cluster 0.

By using 4 for the min_pts parameter, we got the same results as for the value 3. For the value 1, we had each sample in a different cluster. So the right min_pts value for us to use was 2 min_pts. Using 2 min_pts resulted in 991 points marked as noise, 2 points in cluster 0 seen in Figure 11, 2 points in cluster 1 seen in Figure 12 and 5 points in cluster 2, which were the same points as seen in Figure 10.

We also experimented with the appropriate eps value for DBSCAN. We used the K-Nearest Neighbors Algorithm to determine the eps value as follows. We examined the k-distance graph and looked for the ”elbow” point, where the

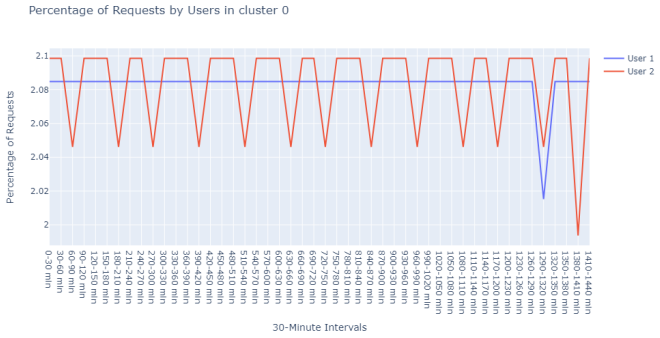

Fig. 11. DBSCAN with 2 min_pts for cluster 0.

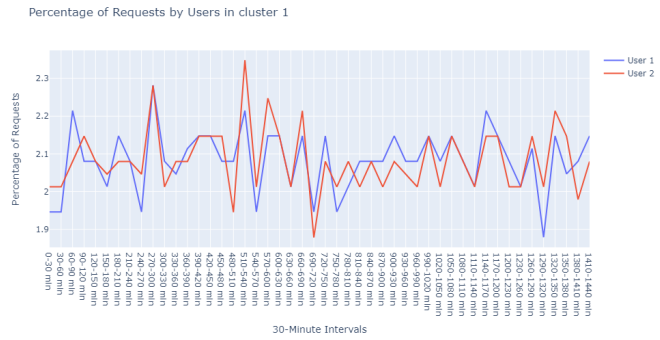

Fig. 12. DBSCAN with 2 min_pts for cluster 1.

plot has the sharpest bend. The distance at this point is considered a good estimate of the eps value for DBSCAN. In our case, the elbow point was at a value y = 8.5 as seen in Figure 13. The eps value 8.5 and the min_pts value 2 resulted in 978 points in cluster 0 and 22 points in cluster -1 (noise). We decided to hand these 22 samples seen in Figure 14 for further analysis to help us gain more insight in our further research.

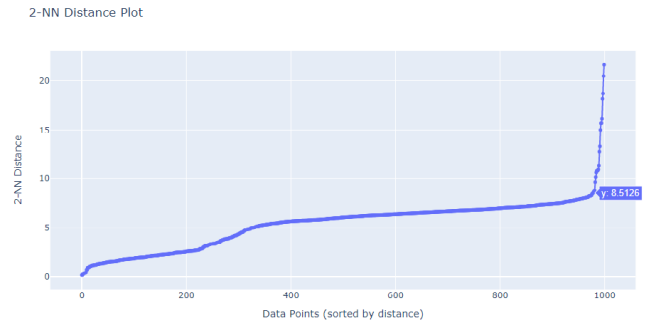

Fig. 13. Determining the eps value for DBSCAN using k-NN.

As a result of clustering anomalous user request patterns from our isolation forest model, we were able to find similarities among different users, which were labeled as anomalous. These results can provide security experts with valuable information they can incorporate into their network monitoring

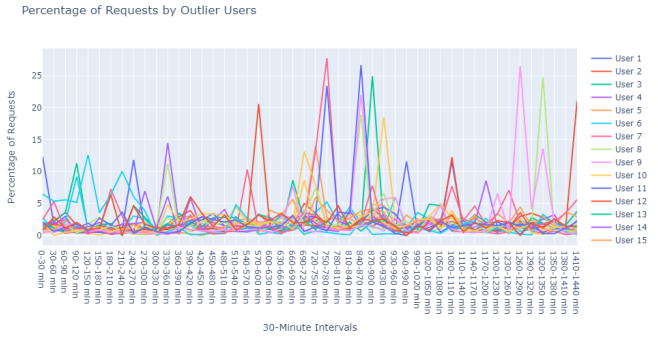

Fig. 14. Noise samples from eps value 8.5 and min_pts value 2.

and take quick action once such pattern occurs.

VIII. Results and Conclusion

By automating the process of identifying outliers in large datasets, network security analysts can benefit from a more effective, accurate, and scalable method of threat analysis. This approach enables them to focus on essential activities, respond to potential threats more quickly, and maintain a proactive stance on network security. Importantly, the parameters utilized in our approach are not specific to NGINX but can be applied to logs from various web server engines, enhancing the versatility and applicability of our methodology. Using isolation forest on an enterprise network dataset, we implemented two methods to identify both broad and specific anomalies, enhancing our understanding of user behavior and aiding in decision-making. The t-SNE technique proved invaluable for visualizing high-dimensional data and identifying clusters based on users' countries and request types. Overall, this approach demonstrates the importance of automation in network security analysis and its potential to provide valuable insights for proactive threat detection and response.

By analysing the behaviours of all different users and their server access patterns, we were able to analyse one day's worth of traffic using the given technique. We clustered user requests based on their request times throughout the day, focusing on the percentage of requests within specific time frames and omitting other attributes to prevent influence from high request values. We attempted various parameters for DBSCAN to find anomalous clusters, which allowed us to identify anomalous request patterns among multiple users, providing security experts with valuable information for network monitoring and enabling them to quickly respond to such patterns.

IX. Future Work

Future work on this paper will focus on creating a framework for quick real-time user anomaly identification based on its findings, as well as further analysing various found clusters to better understand complex user activity. This will allow us to further identify found patterns and create a labeled dataset that will allow us to calculate the accuracy of the model by examining false positives and negatives.

Another problem is that consumers are dispersed over the world, resulting in varied request times because of their time zones, which we will need to address in our further research. Additionally, we'll work to lower the model's CPU usage so that it can be retrained even quicker and be able to detect anomalies in a dataset that is even larger and contains weeks' worth of user behaviour rather than just a single day's worth.

Furthermore, we would like to perform a complete comparison study and identify a suitable strategy for meaningfully comparing our approach method to other studies.

References

[1] L. Benova and L. Hudec, "Detecting anomalous user behavior from NGINX web server logs," in *2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC)*, 2022, pp. 1–6.

[2] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

[3] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, no. 1, pp. 949–961, 2019.

[4] Y. Gao, Y. Ma, and D. Li, "Anomaly detection of malicious users' behaviors for web applications based on web logs," in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, 2017, pp. 1352–1355.

[5] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, "Loglens: A real-time log analysis system," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1052– 1062.

[6] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang et al., "An empirical investigation of practical log anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1404–1415.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ser. SOSP '09*, 2009, pp. 117–132.

[8] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 102–111.

[9] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *USENIX Annual Technical Conference*, 2010, pp.

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, 2017.

[11] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, et al., "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," *IJCAI*, vol. 19, no. 7, pp. 4739–4745, 2019.

[12] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pp. 1573–1582, 2016.

[13] Liu, F.T., Ting, K.M., and Zhou, Z.H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422. IEEE.

[14] Martin Ester, Hans-Peter Kriegel, J"org Sander, Xiaowei Xu, and others. *A density-based algorithm for discovering clusters in large spatial databases with noise.* In *kdd*, volume 96, number 34, pages 226–231, 1996.

[15] Geoffrey E Hinton and Sam Roweis. *Stochastic neighbor embedding. Advances in neural information processing systems*, volume 15, 2002.