Early Detection of Malicious Activity in Log Event Sequences Using Deep Learning

Andrej Ralbovský, Ivan Kotuliak
Slovak University of Technology
Bratislava, Slovakia
{andrej.ralbovsky,ivan.kotuliak}@stuba.sk

***Abstract*—The increasing frequency and severity of security incidents, particularly those involving previously unknown malicious activity, highlight the urgent need for innovative approaches to improve the cybersecurity posture of the organization. Early detection of these activities is essential for preventing security breaches and minimizing data loss. This paper deals with application of deep learning techniques for the detection of malicious activities within network log data. Our methodology involves utilizing logs obtained from prior security incidents detected by Security Information and Event Management (SIEM) to construct a model adept at identifying patterns within user-system connection sequences. In particular, benign activity logs were collected from Elastic stack during periods devoid of reported incidents to establish a baseline for normal activities. Central to our approach is the prioritization of minimizing false positives, a crucial aspect for building trust in the system among security analysts. Our model is able to detect malicious activity by analyzing only about a quarter of the sequence length, compared to detection by SIEM using manually defined detection rules. By reducing the occurrence of irrelevant alerts, our model aims to provide a high-fidelity detection system that empowers analysts to focus their attention on genuine threats and not be subjected to alert fatigue, ultimately enhancing an organization's overall cybersecurity.**

I. Introduction

Every modern organization relies on a vast network of electronic devices and information systems connected to the internal network and the Internet. They typically process significant amounts of information that, due to their value, is at risk of attack. Therefore, organizations must implement appropriate measures as part of risk management to eliminate or minimize the impact of a successful attack. A large portion of these measures are technical in nature.

What many of these devices and systems have in common is their ability to generate logs. A log is a digital record of events from a device or application, like a journal entry [1]. Logs can be collected from existing devices and applications. However, analysing the collected information requires experienced administrators and security specialists [2].

As the number of data sources for analysis grows with addition of new devices and systems, so does the demand on human resources. Due to the shortage of specialists in this area, automating as many processes as possible is crucial for shortening the response time to security incidents.

Artificial intelligence (AI) is a powerful tool for creating automated security solutions. Traditional log analysis relies heavily on manual effort from security specialists, who are increasingly stretched thin due to the ever-growing volume of data generated by modern IT environments.

Anomaly detectors using AI can analyze vast amounts of log data much faster and more efficiently than humans. These models can learn from past security incidents and benign activity logs to identify patterns that might indicate a potential attack. By automating the detection of these patterns, AI empowers security teams to focus their expertise on investigating and responding to genuine threats, ultimately improving an organization's overall cybersecurity.

In our research we propose a method of detecting malicious activity from log data earlier than using traditional means such as manually defined rules in SIEM. We created a model, that can detect malicious activity by using, on average, only about a quarter of the log event sequence, thus enabling rapid reaction on such activity. While rule-based detection systems are widely used, writing rules is time consuming and prone to human errors. Compared to rule-based methods, our model needs just log data of known malicious activity to train on.

A. Log data

Log data is a vital source of information for security and system analysis. It consists of chronological records generated by devices and systems [1]. These records capture specific system states at a particular point in time, serving as a valuable resource for debugging and forensic analysis in case of failures, security incidents, or unexpected behavior. Logs typically exist in textual form and can vary in structure – from unstructured messages written in natural language (e.g., "File transfer failed due to network timeout.") to structured objects with attributes represented by key-value pairs (e.g., rule_action=Accept srcPort=41272).

Logs therefore require parsing to extract meaningful information [3]. While some parsing tools focus solely on extracting attributes from individual messages, others categorize them into these broader event types. These event types are also referred to as log keys, log signatures, log events, or simply events. For consistency, this paper will simply refer to them as *events*. Depending on the structure of the log, log type can be decided by single attribute, multiple attributes for structures objects or by identifying constant and variable parts in unstructured messages. Despite the lack of a universal log format, a timestamp indicating when the log occurred can

usually be extracted. This timestamp is a key attribute when it comes to extracting sequences of events for analysis [4].

Popular datasets used in anomaly detection evaluation, such as HDFS and BGL, consist of system logs. However, our approach necessitates different log data to learn from past security events and identify patterns in user-system interactions. Therefore, we created our own dataset in collaboration with the GOV CERT SK unit of the National Agency for Network and Electronic Services. The data obtained is sensitive and subject to confidentiality.

B. Anomaly detection

Anomaly detection focuses on identifying instances in a dataset that exhibit rare or unexpected characteristics, making them stand out from the rest of the data. These instances are often referred to as anomalies. When analyzing independent data, identifying anomalies is often a matter of spotting outliers. Outliers, also known as point anomalies, are individual data points or small groups of data points that deviate significantly from the majority of the data. Ordered data, where data points are interdependent, presents two additional anomaly types: contextual and collective [1]. Contextual anomalies appear benign individually but can become malicious due to their surrounding context, especially the timing. Imagine a user login that usually happens in the morning and during the day, but once the user logs in at midnight. This deviation from the expected timing makes the event a contextual anomaly. Collective anomalies, on the other hand, involve groups of events that aren't individually suspicious. However, when these seemingly benign events occur together in a specific sequence, they might collectively indicate a potential issue. For instance, in log data analysis, a sequence of ordinary logs might collectively point towards a malicious activity.

Existing log anomaly detection approaches can be categorized along two dimensions: the type of anomaly (sequential or quantitative) and the learning approach (supervised or unsupervised) [5] [6]. A core assumption for many anomaly detection techniques is a significant class imbalance, where normal data instances far outnumber anomalies. And since anomaly labels are often scarce in real-world applications, unsupervised methods receive more attention. However, the common drawback of unsupervised learning is that it can result in a high false positive rate by learning patterns of normal logs and considering everything else as anomalous behavior. This can make unsupervised approaches unsuitable for real-world deployments prone to alert fatigue. On the other hand, supervised approaches for anomaly detection are relatively rare due to the challenge of acquiring sufficient labeled anomaly data.

C. Related work

This section provides a brief overview of existing research related to deep learning for anomaly detection in log data. Deep learning has emerged as a powerful approach, overcoming limitations of traditional machine learning methods like inflexibility, inefficiency, and weak adaptability. This section explores several prominent deep learning architectures used for this purpose.

1) DeepLog: utilizes a Long Short-Term Memory (LSTM) model to capture sequential relationships between log events [3]. It predicts the next event based on the preceding sequence and flags deviations from predictions as anomalies. This approach achieved an F1 score of 0.96 on the HDFS dataset.

2) LogAnomaly: also employs an LSTM model but utilizes log count (quantitative) vectors as input [5]. Additionally, it introduces "template2vec" a method for representing log templates as semantic vectors based on synonyms and antonyms. Similar to DeepLog, it predicts the next log event and considers deviations as anomalies. LogAnomaly achieves F1 score of 0.95 and 0.96 on the HDFS and BGL datasets, respectively.

3) PLELog: tackles the challenge of limited labeled data by employing probabilistic label estimation alongside an attention-based Gated Recurrent Unit (GRU) neural network for anomaly detection [7]. This approach classifies log sequences as normal or abnormal and outperforms existing semi-supervised methods, achieving F1 score of 0.96 and 0.98 on HDFS and BGL datasets, respectively.

4) LogRobust: incorporates pre-trained word embedding models (FastText) with TF-IDF weights to represent log templates. These vector representations are then fed into an Attention-based Bi-LSTM model for anomaly detection [8]. This approach demonstrates robustness against unstable log events and achieves an F1 score of 0.99 on the original HDFS dataset and 0.89-0.96 on synthetic datasets.

5) Convolutional Neural Networks: has been shown to be a feasible approach for log anomaly detection [9]. Logs are grouped into sessions and transformed into a trainable matrix, which is fed into a CNN model to classify log sequences as normal or abnormal. This approach achieves an F1 score of 0.98 on the HDFS dataset.

6) LogBERT: is based on transformer architecture. Despite its name it does not incorporate BERT language model, authors were just inspired by it and the model is trained from scratch [10]. This approach achieved an F1 score of 96.64 on Thunderbird dataset.

7) LogFit: uses fine-tuned BERT model to detect anomalies in logs [11]. It achieved precision of 99.78 and F1 score 94.97 on HDFS dataset.

II. Methodology

Our research benefited from access to sources containing logs from both malicious and benign activities. This unique resource allowed us to develop a novel approach for training anomaly detection models. Rather than employing traditional anomaly detection techniques, we opted to train a model that can recognize patterns indicative of malicious activity based on previous security incidents. This approach acknowledges the inherent nature of cyberattacks, which often progress through various phases, with some not being reflected in log data [12]. Nonetheless, any captured logs associated with security incidents provide valuable training material. Consequently, we

categorized logs from past investigations under the umbrella term ”malicious activity” to train our model on recognizing these specific patterns.

A. Data collection

To create a comprehensive dataset, we collected logs from two primary sources: Elastic stack and SIEM. We focus on logs from IPS – representing the entry point for potential attackers into the organizational network.

1) Benign activity: To obtain logs for benign activities, we queried Elastic stack via its API after verifying through SIEM that no security incidents occurred during the corresponding time period. Since Elastic stack stores parsed logs, syntactic analysis was unnecessary. However, working with parsed logs introduced the additional step of associating each log with a template. This mapping requires manually defining attributes, identified through exploratory data analysis, that can be used to pair logs with their corresponding template. These attributes vary depending on the log type.

2) Malicious activity: To obtain data on malicious activities, we directly extracted logs from security incidents identified and confirmed (marked as true positives) within the SIEM system. This targeted selection ensures the model trains on malicious behavior. However, it’s important to acknowledge that not all security incidents exhibit sequential patterns (such as identifying signatures). SIEM, by design, aggregates logs from diverse sources, which means that logs have a wide range of attributes. To address this, we filtered the IPS logs that have a well-defined set of common attributes. Furthermore, we opted to extract these logs in their raw format, allowing for feature engineering during the data preprocessing stage. This stage involved extracting critical features like actions and IP addresses, ultimately transforming the raw data into a format suitable for model training.

B. Data cleaning and preprocessing

For data cleaning and preprocessing, we perform several steps (as illustrated in Fig. 1). First, we parse the logs to extract relevant attributes such as timestamps, source and destination IPs, and potentially other relevant information. This process transforms the raw logs into a structured format suitable for further analysis. Next, mapping is required to group related events together. Finally, a grouping strategy is chosen, and valid features are extracted for building the detection model.

This approach is common in deep learning for anomaly detection applied to log data [3], [5], [7]. Extracting informative features that incorporate contextual information can further enhance model explainability.

1) Parsing: Once logs are collected from the two sources, they undergo preprocessing. This initial stage begins with an organized set of logs, denoted as $l_1, l_2, l_3, ...$ Portion of these logs requires parsing, a process that extracts relevant information and transforms each log pl_i for each log l_i. Parsed logs typically contain key-value pairs representing extracted parameters, structured like $[k_i : v_i, k_j : v_j, ...]$.

Fig. 1. Preprocessing steps for sample firewall log

TABLE I
MAPPING BETWEEN LOGS AND EVENTS BASED ON THE ACTION ATTRIBUTE.

Action	Event
Drop	E0
Deny	E1
Accept	E2
Prevent	E3
Detect	E4
Bypass	E5

2) Mapping: This step focuses on extracting events from logs. An event represents the type of log. The parsed log (pl_i) is therefore mapped to one of events (E_j). During our experiments we identified relevant attributes to the type of log as typical actions with values like ”Allow,” ”Deny,” and ”Inline.” etc. Illustration of the principle of mapping between logs and their corresponding events is shown in Table I.

3) Grouping: The subsequent step involves logical organization of events into groups for individual analysis. There are several common strategies for grouping logs, each with its own advantages and disadvantages. These are:

- Fixed-length window – Logs are grouped in the order they were generated into windows with a fixed length. For example every 100 consecutive logs are considered a window. This strategy is simple to implement and interpret and has very good performance. However, the disadvantage is that it’s not suitable for every dataset (such as ours) and can also lead to information loss.
- Sliding window – Logs are grouped in the order they were generated into fixed windows with a step size. Step size defines the shift between consecutive windows, in other words, their overlap. This strategy can eliminate information loss; however, like the previous one, it is also not suitable for every dataset.
- Session-based window – Logs are grouped based on

unique session identifier and it is not bound by time intervals or predefined sizes. This results in windows of varying lengths corresponding to duration of individual sessions. It can be computationally expensive, but enables precise analysis of behaviour for individual flows.

In our approach, we decided to group logs based on IP address. This grouping allows for individual analysis of user connections based on a unique identifier. From each group of events (parsed logs related to a specific IP), event sequences can be extracted. These sequences represent the chronological order of events associated with a particular user.

To minimize information loss for individual connections, we utilize sliding windows with a constant step of a single log. This ensures thorough analysis for each connection while retaining identified session boundaries [3].

C. Feature extraction

Feature extraction involves transforming pre-processed events into numerical representations suitable for anomaly detection models. We considered primarily sequential and quantitative vectors:

- Sequential vectors capture the sequential patterns within event sequences. They preserve the temporal relationships between events by encoding their order and timing. This approach relies on the assumption that legitimate program execution or user activity follows consistent event sequences and deviations from these patterns might indicate anomalies [5].
- Quantitative vectors focus on extracting quantitative attributes from events. It leverages the assumption that certain statistical properties remain consistent during normal program execution [5]. Features might include counts of specific event types, along with relevant statistical information like means and variances.

In our approach, we primarily focused on extracting sequential vectors and a combination of sequential and quantitative vectors. We acknowledge that while quantitative vectors offer insights into statistical properties of logs, our experiments indicated limited additional effectiveness in detecting malicious activities. This suggests that the effectiveness of feature extraction techniques can depend on the specific characteristics of dataset or mapping granularity.

D. Development of model

The detection of malicious activity is defined as a multi-class classification problem, where all events E serve as classes. The model is trained to predict the upcoming event based on the input sequence of preceding events. The actual event is compared with the most probable candidates for the next event template obtained using the probability distribution function. Our approach is based on [3], which has been shown to be suitable for detecting sequential anomalies, but instead of training on normal non-anomalous sequences, we trained it on sequences containing malicious activity.

The input to the model is a vector which contains the last h events.

$$w = E_{i-h}, E_{i-h+1}, ..., E_{i-2}, E_{i-1} \quad (1)$$

This is used to predict the next event E_i. The output of the model is the probability distribution

$$Pr(E_i \ = k_i|w) \quad (2)$$

for all possible events E. The threshold of how many most probable candidates to consider is determined by the parameter we denote as k. This parameter represents the number of most probable events with which the actual event is compared. If the actual event does not match any of the predicted events, the sequence is evaluated as malicious activity.

III. Evaluation

A. Dataset

In our experiment we use a dataset containing a total of 17 272 630 logs. These logs are labelled into two classes: malicious activity and benign activity. The dataset breakdown is as follows:

- Malicious activity: 4 550 022 logs (26.34% of the dataset)
- Benign activity: 12 722 608 logs (73.66% of the dataset)

The dataset was collected in partnership with GOV CERT SK unit of the National Agency for Network and Electronic Services. They were collected from two systems – SIEM and Elastic stack.

We adopted a following splitting strategy. We specifically split the malicious logs into two sets: 80% designated for training and 20% reserved for validation. The training set allows the model to learn the patterns present within normal malicious session sequences. The validation set serves to monitor the model's performance on unseen malicious data during the training process, helping to prevent overfitting. The remaining portion of the dataset, consisting entirely of benign sessions is used for testing. This approach ensures a thorough evaluation of the model's ability to differentiate benign activity from potentially anomalous malicious sessions during the final assessment.

B. Models

We experimented with two anomaly detection models inspired by DeepLog [3] and LogAnomaly [5]. From DeepLog, we adopted the concept of using sequential vectors to capture the order of events within a session. Similarly, inspired by LogAnomaly, we investigated combining both sequential and quantitative vectors in feature extraction. DeepLog and LogAnomaly were originally designed to detect anomalies in program execution paths within system logs [3]. In our approach, we adapted these models to a different task: finding similarities between IPS logs and previous security incidents to identify malicious activity.

We also experimented with RobustLog [8], which utilizes semantic vectors and relies on NLP. However, we discovered that our dataset, containing structured objects with attributes, is not a suitable data structure for incorporating semantic vectors.

For our experiments, we built upon LogDeep [13], a publicly available reimplementation of both DeepLog and LogAnomaly models. Utilizing LogDeep as a foundation allowed us to efficiently leverage these approaches due to its existing implementation. We made necessary modifications to LogDeep to accommodate our specific data format and experiment. First, we unified the input format for both training and prediction stages and replaced hardcoded values with parameters, streamlining the workflow and improving efficiency. Second, we enhanced the contextual information provided during prediction by incorporating timestamps, session IPs and related logs. We also added a softmax function to obtain probability distribution for each predicted event. All these things matter to security analysts when responding to threat and it allows for better interpretability of the model's outputs, which helps operators prioritize potential security incidents. Finally, we extended the set of evaluated metrics to include EDR (explained in section III-D2), specificity and accuracy to gain deeper insights into the model's performance in the context of malicious activity detection of security incidents within IPS logs. These modifications not only enhance the interpretability of the model's outputs but also lay the groundwork for establishing a future feedback loop. Additionally, these changes support the implementation of online detection capability for real-time analysis of IPS logs.

It's important to note that while we drew inspiration from DeepLog and LogAnomaly, we did not directly modify their core architectures. We explored alternative approaches such as attention networks and BiLSTMs (Bidirectional Long Short-Term Memory) to potentially enhance anomaly detection. However, in our experiments, these approaches did not provide any significant additional value compared to the core functionalities offered by DeepLog and LogAnomaly, as implemented in LogDeep.

C. Parameters

There are two important parameters to consider when configuring our anomaly detection models – window size h and most probable candidates k [3].

The window size h parameter determines the number of preceding events considered to predict the next event in a session. This parameter influences the model's ability to capture temporal patterns within session data. A larger window size means that we will discard shorter sessions until they reach required amount of events.

Most probable candidates k represent the most probable events identified by the model for the next event in the sequence. The selection of the appropriate number of candidates to consider for detection is crucial.

In our approach, we utilize a constant step size of 1 for the sliding window within each session. This minimizes information loss by ensuring that every single event within a session is considered by the model.

D. Evaluation metrics

1) Core evaluation metrics: Our core metrics are based on sessions (not on individual windows within sessions). We define core evaluation metrics as following:

True Positive (TP) represents a session that was correctly classified as malicious. This occurs when the model identifies any window within the session as malicious, consequently classifying the entire session as malicious.

False Positive (FP) represents a benign session that was incorrectly classified as malicious. This happens when the model identifies a window within a benign session as anomalous, leading to the entire session being misclassified as malicious.

True Negative (TN) represents a benign session that the model correctly classified as normal. This signifies the model's ability to identify entirely benign sessions without raising false alarms.

False Negative (FN) represents a malicious session that the model missed and classified as normal. This occurs when the model fails to identify any anomalous windows within a malicious session, resulting in the entire session being misclassified as benign.

2) Derived evaluation metrics: We also calculate derived evaluation metrics, such as precision, recall, specificity, F1 score and accuracy.

Since we are working with sessions that were previously captured, we also propose another metric, EDR. Early Detection Rate measures the proportion of the sequence length (denoted by SL) that the model needs to analyze (denoted by A) before making an accurate prediction. In other words, it indicates how much of the sequence the model needs to see on average to correctly detect malicious activity.

$$EDR = \frac{A}{SL} \tag{3}$$

To calculate the EDR for the entire dataset, we first compute the EDR for each TP instance. Then, we sum these individual EDR values for all TP instances and divide by the total number of TP instances.

$$EDR_{\text{dataset}} = \frac{\sum_{i=1}^{TP} EDR_i}{TP} \tag{4}$$

Where:

- EDR_{dataset} is the Early Detection Rate for the entire dataset.
- EDR_i is the Early Detection Rate for the i-th true positive instance.
- TP is the total number of true positive instances.

Precision reflects the proportion of identified sessions that were truly malicious. In simpler terms, it tells us how much we can trust the model's positive predictions (malicious activity). A high precision indicates that the model is generating a low number of false alarms and mostly identifying real malicious activities.

$$Precision = \frac{TP}{(TP + FP)} \tag{5}$$

Recall (sensitivity) reflects the model's ability to identify malicious activity. It tells us how good the model is at

capturing all the malicious activities present in the data. A high recall indicates that the model is successfully identifying most of the attacks, minimizing the number of missed threats.

$$Recall = \frac{TP}{(TP + FN)} \tag{6}$$

Specificity focuses on the model's effectiveness in correctly classifying benign activity. It tells us how good the model is at avoiding false alarms. A high specificity indicates that the model is not prone to misclassifying benign activity as malicious, reducing unnecessary investigations.

$$Specificity = \frac{TN}{(TN + FP)} \tag{7}$$

The F1 score combines Precision and Recall into a single score, providing a balanced view of both metrics. It considers both the model's ability to identify true positives while avoiding false positives. A high F1-score indicates that the model is performing well in terms of both precision and recall.

$$F1\ score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \tag{8}$$

Accuracy measures the overall proportion of correctly classified sessions. It tells us how often the model makes the right prediction, regardless of class (malicious or benign). While a high accuracy might seem ideal, it can be misleading in imbalanced datasets.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

E. Environmental setting

Our experiments are conducted on a server with Intel® Core™ i7-9700 CPU @ 4.70 GHz 8 cores CPU with 32GB memory. All experiments were performed on CPU.

IV. Results and Discussion

In this section, we present the results obtained from our experiments with the proposed models on the dataset described in the previous chapter. We begin by outlining the process of generating sequences and the configuration of the models used. Subsequently, we delve into a detailed analysis of the experimental results, focusing on the performance metrics and comparisons between different model variations.

A total of 202 723 sequences were constructed from the dataset. Among these, only around 12 000 are used in experiments depending on minimum window size. Malicious activity was represented in around 83% sequences. We ended up using 8000 sequences for training and remaining 2000 for validation of malicious activity and 2000 for testing benign activity.

We first tried to estabilish a baseline for what an optimal window size might be. As shown in Fig. 2, models tended to perform better with smaller window size, but that can be a side effect of removing sequences shorter than minimum window size, thus changing ratio for classes. In general, models' performance was very similar.

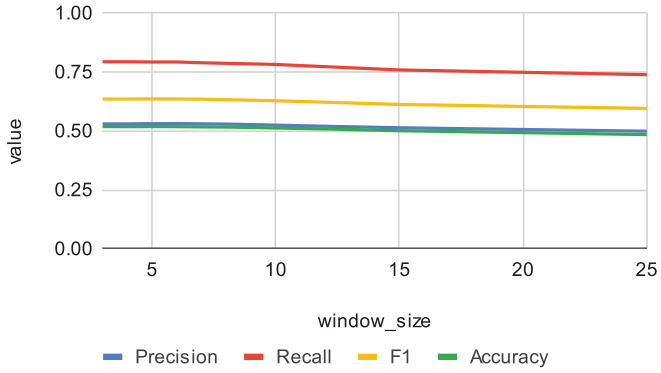

Fig. 2. Model performance with different window sizes. [sequential, k = 1]

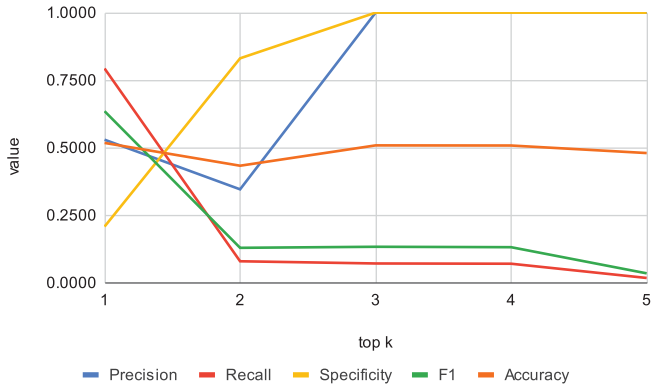

Fig. 3. Model performance with different top k. [sequential, window size = 4]

Having established a baseline, we experimented with different top k values for next event prediction. As shown in the plot in Fig. 3, using lower values for k results in very high false positives (manifested as low specificity). Even with $k = 2$ the specificity remains low. However using $k = 3$ results in a model that can very well identify benign activity and achieves perfect precision and specificity. The trade-off for this is a very low recall, meaning the model misses a lot of malicious activity.

In a production environment, operators are prone to alert fatigue. Models that generate excessive false positives will erode their trust in the system over time. From this perspective, false positives are highly undesirable. Additionally, malicious activity in this dataset originates from alerts created by integrated security systems and devices. Therefore, for this model to detect everything, it would only make sense in an edge case where it's the sole security appliance in the entire network. But our model is just additional method of detection, so any missed malicious activity shall be detected by SIEM, albeit later. Finally, a security incident doesn't inherently guarantee a sequential pattern that this model can capture. Determining such patterns would require a non-trivial amount of time, such

TABLE II
BEST MODEL PERFORMANCE ACROSS DIFFERENT FEATURES AND WINDOW SIZES.

Features	Window size	Top k	Precision	Recall	Specificity	F1	Accuracy	EDR	Time [s]
seq	3	3	1.0000	0.0725	1.0000	0.1352	0.5098	0.21	540
seq	4	3	1.0000	0.0714	1.0000	0.1332	0.5085	0.25	540
seq+quan	4	3	1.0000	0.0714	1.0000	0.1332	0.5085	0.23	960
seq+quan	5	3	1.0000	0.0709	1.0000	0.1324	0.5074	0.26	1020
seq	5	3	1.0000	0.0705	1.0000	0.1316	0.5071	0.19	584
quan	15	4	1.0000	0.0361	1.0000	0.0697	0.4992	0.46	640

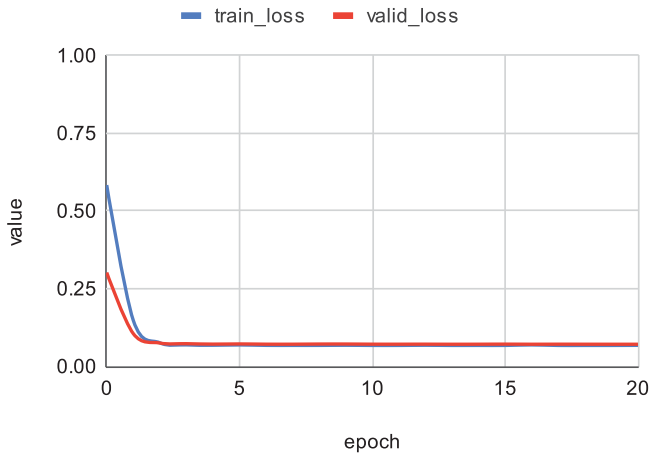

Fig. 4. Model loss curve during first 20 training epochs. [sequential, window size = 4]

as hours of manual investigation. Because of these factors, we believe that a low False Positive Rate (FPR) is significantly more important than a high recall.

Our experiments revealed that the model learns sequential patterns very quickly within a few epochs, as shown in Fig. 4. This rapid convergence could indicate either low event granularity in the data or a dataset with limited complexity. Each training epoch takes approximately 3 minutes on average, with larger window sizes leading to longer training times. Consequently, training a model with window size 4 for 30 epochs requires roughly 1.5 hours.

To explore the impact of different feature sets, we trained models using both sequential and quantitative features. Summary of their performance is shown in Table II. Overall, the performances are very similar when using sequential (DeepLog) and a combination of sequential and quantitative (LogAnomaly) features. The most significant difference lies in processing time – using both features requires roughly double the processing duration. For instance, predicting 4,000 sequences with a window size of 4 takes approximately 960 seconds for the combined features model, compared to 540 seconds for the sequential-only model. We also trained a model using solely quantitative features, which exhibited slightly lower performance compared to the other approaches.

Afterwards, we investigated the Early Detection Rate (EDR) of these models. The results were promising, with the models achieving an average EDR between 0.19 and 0.46. This translates to detecting malicious activity within the first quarter of the sequence length. In simpler terms, the models can identify suspicious behavior relatively early during an attack phase, allowing for a faster response and potentially mitigating the entire cyberattack. This early detection capability is particularly valuable when it comes to security, where prompt intervention can significantly minimize damage.

V. CONCLUSION

This research explored a novel approach for training deep learning anomaly detection models by leveraging access to logs from past security incidents. These logs, categorized as ”malicious activity” provided valuable data to train a model that recognizes patterns indicative of such activities. This approach acknowledges the limitations of log data, as it may be incomplete or lack specific details about attackers’ other actions.

The successful training of the deep learning model demonstrates the viability of this approach. The results suggest that the model can effectively detect malicious activity with a focus on minimizing false positives, even achieving early detection within attack sequences requiring just as little as quarter of the sequence. However, the cost for minimizing false-positives is a significant decrease in recall to less than 10%. But whereas our method is just additional detection system alongside SIEM, specificity metric is much more important than recall.

Building upon this research, further exploration into advanced data analysis techniques could be beneficial. This could involve digging deeper into related logs, such as system logs or web server logs, to analyze user behavior patterns. By examining how users interact with different systems under various circumstances (e.g. during a potential attack or under normal operation), we can potentially gain valuable insights into attacker behavior. This deeper understanding can further bolster our deep learning detection model, leading to improved accuracy and better real-world applicability.

In future work, we would like to evaluate the efficiency of the model in detecting malicious activities it was not trained on.

ACKNOWLEDGMENT

This research was supported by project KEGA No. 004ZU-4/2024 ”Improving the quality of education in the field of cyber security”.

REFERENCES

[1] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.mlwa.2023.100470

[2] Q. Wang, X. Zhang, X. Wang, and Z. Cao, "Log sequence anomaly detection method based on contrastive adversarial training and dual feature extraction," *Entropy*, vol. 24, no. 1, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/1/69

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[4] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 92–103.

[5] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI'19. AAAI Press, 2019, p. 4739–4745.

[6] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: how far are we?" in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1356–1367. [Online]. Available: https://doi.org/10.1145/3510003.3510155

[7] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1448–1460.

[8] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817. [Online]. Available: https://doi.org/10.1145/3338906.3338931

[9] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 151–158.

[10] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[11] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Logfit: Log anomaly detection using fine-tuned language models," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1715–1723, 2024.

[12] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide," 2012-08-06 2012.

[13] D. Lee, "Logdeep," Web: https://github.com/d0ng1ee/logdeep, 2020.