Towards Detecting Anomalies in Log-Event Sequences with Deep Learning: Open Research Challenges

Patrick Himler, Max Landauer, Florian Skopik, Markus Wurzenberger
Austrian Institute of Technology, Center for Digital Safety and Security
Giefinggasse 4, 1210 Vienna, Austria
firstname.lastname@ait.ac.at

ABSTRACT

Anomaly Detection (AD) is an important area to reliably detect malicious behavior and attacks on computer systems. Log data is a rich source of information about systems and thus provides a suitable input for AD. With the sheer amount of log data available today, Machine Learning (ML) and its further development Deep Learning (DL) have been applied for years to create models for AD. Especially when processing complex log data, DL is often able to achieve better performance than ML. To detect anomalous patterns that span over multiple log lines, it is necessary to group these log lines into log-event sequences. This work uses a Long Short-Term Memory (LSTM) model for AD which is one of the most important approaches to represent long-range temporal dependencies in log-event sequences of arbitrary length. This means that we use past information to predict whether future events are normal or anomalous. For the LSTM model we adapt a state of the art open source implementation called LogDeep. For the evaluation, we use a Hadoop Distributed File System (HDFS) data set, which is well studied in current research, and an open source Audit data set provided by the Austrian Institute of Technology (AIT). In this paper we show that without padding, a common preprocessing step used that strongly influences the AD process and artificially improves detection results and thus accuracy in lab testing, it is not possible to achieve the same high quality of results shown in literature. Furthermore, we analyze limitations of DL approaches applied for AD and list future research priorities and design challenges.

CCS CONCEPTS

• **Security and privacy** → *Intrusion/anomaly detection and malware mitigation.*

KEYWORDS

log event sequences, anomaly detection, deep learning, lstm

ACM Reference Format:
Patrick Himler, Max Landauer, Florian Skopik, Markus Wurzenberger. 2023. Towards Detecting Anomalies in Log-Event Sequences with Deep Learning: Open Research Challenges. In *European Interdisciplinary Cybersecurity Conference (EICC 2023), June 14–15, 2023, Stavanger, Norway.* ACM, Stavanger, Norway, 7 pages. https://doi.org/10.1145/3590777.3590789

1 INTRODUCTION

AD has been an actively researched field for decades in various domains [4]. The goal of AD is to find events and event sequences that deviate from normal behavior as efficiently and timely as possible [3]. In this paper, we investigate the detection of anomalies in the cyber security domain using log data analysis. In recent years, cyber attacks have become more sophisticated and the attack surface has tremendously increased because of nowadays complex computer systems and networks. AD is used in Instrusion Detection Systems (IDS) to automatically detect and classify intrusions, attacks, or violations of security policies in infrastructures at network-level and host-level [23]. Conventional signature-based IDS, using patterns of already known attacks and malicious behavior, have become insufficient and the need for more adaptability, to enable detection of unknown attacks such as zero-day exploits, is immanent. As a result, we see a shift towards anomaly based IDS, which use various data sources like textual log data and network traffic data to reliably discover deviations from a desired system behavior.AD supports reaching the goals of minimal maintenance, human interaction, and delay in response time [25].

Currently, especially DL, a branch of highly performant ML algorithms, is a hot spot in AD research. DL aims to discover the essential differences between normal and abnormal data with high accuracy [16]. DL supports security experts to react quickly and effectively to known and unknown attacks, and to gain a broad overview of the threat landscape. It is also useful to utilize log data for training DL models. Log data provide more detailed insights for behavioural understanding of a system than network traffic data. Contrarily to many available state of the art AD systems, which process network traffic data, a paradigm shift towards log data should be considered, because log data is generally available in unencrypted form and contains low-level traces of system activities [25]. Most of the time, working with log data requires preprocessing steps, where the textual log data are parsed into numeric formats. After parsing, a DL model can process log data and classify them into normal and anomalous log events [11]. If we want to detect not only individual anomalous log lines, i.e. point anomalies, but also contextual anomalies, several log lines are combined into log-event sequences. If parts of the data set are anomalies only in a certain context but not isolated from that context, we speak of contextual anomalies. To detect contextual anomalies, we need to consider both contextual attributes, like passed time between data instances and behavioral attributes such as occurrences of specific data instances [21]. The predestined DL algorithm here is LSTM because it uses past information to predict whether future events are normal or anomalous.

Our contributions focus on in pointing out limitations when working with log-event sequences and DL of currently used approaches and future research directions. The contributions are as follows:

- A critical discussion of state of the art approaches.
- An approach for detecting anomalies in HDFS- and Audit log-event sequences using DL.
- A discussion of current limitations and outlook for future research topics in this area.

The remainder is structured as follows: Section 2 gives an overview of state of the art approaches for log-based AD using DL models. Next, Sect. 3 introduces our chosen approach describing all necessary steps from raw log data to AD. Section 4 evaluates our approach and compares it with state of the art solution [8]. Important limitations are critically discussed and open research challenges are listed in Sect. 5. Finally, Sect. 6 concludes this paper.

2 BACKGROUND AND RELATED WORK

LSTM models can be classified among others into supervised, semi-supervised and unsupervised based on the need for data labels during training. Labels indicate whether a respective data instance is normal or anomalous. The differences between the individual approaches are as follows [13][4][22]:

- Supervised: A fully labeled training set containing both normal and anomalous data is required. A common approach is to build a predictive model for both data classes. Then unseen data instances are tested with the model to determine which class they belong to. At first glance, it is easy to create a model like this, but it has two major disadvantages: First, usually data sets contain fewer anomalies than normal data which leads to an imbalanced class distribution. Second, it is not trivial and thus challenging to label the anomaly class.

- Semi-supervised: The prerequisite is that the training set only contains normal data. After training with a portion of normal data, the resulting model is corresponding to normal expected behaviour and can then be used to identify anomalies in the test data set. This technique assumes availability of normal training data sets which can be challenging in some application domains.

- Unsupervised: The system learns independently to distinguish between normal and anomalous data without the prerequisite of a labeled training set. This approach makes use of the intrinsic properties of data instances. In principle, it is often the case that investigated data sets have fewer anomalies than normal data. The trained model should be robust against those few anomalies. If this assumption does not apply, it will lead to a high false alarm rate.

We have chosen to look into semi-supervised approaches for this paper, because they reflect a realistic scenario with regard to real world applications to first learn a model in an anomaly-free area with normal data and then switch to live operation. This is also reflected in the assumption that anomalies are predominantly rare and sometimes difficult to capture as opposed to normal data [22].

In recent years, many DL models to analyze log data and detect anomalies have been proposed [27][9][18]. An in-depth research was performed. To narrow down the results, we filtered by the number of citations, the year of publication and the data sets used. After filtering, the three approaches described in the following sections stood out.

2.1 DeepLog

The authors of [8] have developed an approach called DeepLog that uses an LSTM as DL model that processes log lines in sequences. The model thereby learns log patterns during normal execution and detects anomalies when log patterns deviate from the trained model. Even though this approach was published in 2017, it still has great significance to this day. As a starting point of development, the authors describe that log entries in most cases have fixed patterns and also follow grammar rules. But they also state that it is still difficult to make a generalization about interesting features for different data sets. To evaluate their implementation the authors use the HDFS data set [26] and the OpenStack data set [8]. The HDFS data set consists of log lines from a primary data storage system. The OpenStack data set contains administrative logs of virtual machine instances. DeepLog works in a semi-supervised way. Therefore only anomaly-free sequences are used for training the LSTM model. First, the Spell [7] parser extracts so-called log keys (=constant part) and parameter values (=variable part) from each log line. This way two separate AD systems can be set up. First, DeepLog verifies if the log key to be examined is a known one. In case it is, it can then check if parameter values indicate anomalies. The sequence in which log keys occur can be used to detect so-called execution path anomalies. This type of anomalies corresponds to the contextual anomalies explained above. The LSTM model uses a set of log keys with a fixed size, called window size, and uses the gathered knowledge to predict which log key should follow. After training the LSTM model, DeepLog outputs possible candidates that were predicted with their respective probabilities. In contrast to this, parameter values are used to find irregularities in log lines with the same log keys. A matrix is built up where each column corresponds to a log key and the corresponding parameter values are entered in the rows. For the evaluation of the matrix, an LSTM model can also be used. The individual parameter values are used as input in the order in which they occur and an attempt is made to generate a prediction for the following parameter value based on this existing history. The structure of the DeepLog architecture described here is shown in Fig. 1.

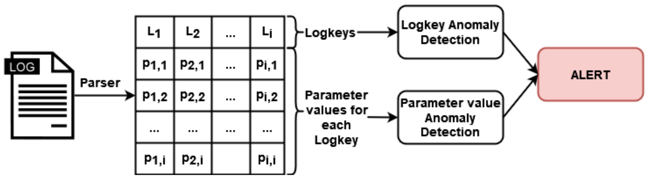

Figure 1: DeepLog architecture based on [8].

The two most important input parameters DeepLog needs are length of the window under consideration and number of top candidates for prediction. The choice of these parameters depends on the problem at hand. For example, if we choose the window size too large, we get a wide view in the past and a better overall picture, but we also have to expect performance losses resulting in long training time. The choice for the number of candidates results in a trade off between AD rate and false alarm rate.

2.2 LogAnomaly

LogAnomaly follows a similar approach as DeepLog. The authors of [17] claim that if we just look at log keys rigidly, one receives many false alarms, because log line structures and dependencies can be highly complex. Therefore they analyze not only log keys but also semantics of the logs. For this, they use a method they call template2vec. This method extracts semantics including synonyms and antonyms. They use LSTM as DL model to predict consecutive logs and HDFS as data set for evaluation. In addition, they also consider the Blue Gene/L (BGL) data set [19], which consists of logs from a supercomputer and was manually labeled. The authors designate LogAnomaly as an unsupervised approach, but they use labels of the data sets as groundtruth for evaluation. According to the experiments performed, LogAnomaly performs better than DeepLog based on the evaluation metrics for the data sets investigated. However, a described case study in this paper shows that DeepLog triggered an alarm faster than LogAnomaly in a single anomaly case scenario.

2.3 LogRobust

Zhang et al. [28] draw attention to instability of log data. Instability is caused by the evolution of the logging process itself and of processing noise in log data. Evolution is based on constant development of software and associated changes in source code and logging statements. The introduction of noise already happens during data collection. But logs can also be misinterpreted during parsing, which both reduces the accuracy of an AD system. To counteract these adverse influences, the authors of LogRobust rely on semantic vectors. Like LogAnomaly, semantic properties of log lines are extracted, but in contrast to previously discussed approaches, LogRobust can also process new log lines if they are similar to known ones. To test this, slightly modified log lines are inserted in the HDFS data set to be examined. It is important to note that training of the model is still performed with unmodified log lines, i.e., the original HDFS training data set. LogRobust also uses log-event sequences as input for a LSTM, but it is a supervised approach and uses 6000 normal log-event sequences and 6000 malicious log-event sequences which are chosen randomly from the HDFS data set to train the model. As expected due to the supervised approach metrics after training are better, for testing without modified log lines, than for the other two approaches shown above. Further evaluation shows that LogRobust still achieves good metrics even with high injection rates of modified log lines in the test data. The authors have not published the source code for their work, but re-implementations are available.

3 APPROACH FOR ANOMALY DETECTION WITH LOG-EVENT SEQUENCES

In this section we present our approach for AD analyzing log-event sequences. Our approach has been tested with the HDFS data set [26] and also an Audit data set [15]. To use the Audit data set some preprocessing steps are necessary which we will explain in this section.

3.1 HDFS Data set

The HDFS is a file system designed for storing large files, batch processing, and to run on commodity hardware. The data set was generated 2009, in a private cloud environment (Amazon's Elastic Compute Cloud) using benchmark workloads and is described in detail in [26]. It consists of 11.2 million system log entries. It was manually labeled by Hadoop domain experts, to identify anomalies, where the anomalies describe incorrect execution paths. 2.9% of all system log entries were labeled as anomalous by those experts. The raw HDFS logs are semi-structured and consist of a header- and a content part. The log data are sliced into sequences according to block_ID's. Then each trace associated with a specific block_ID is assigned a ground truth label: normal/anomaly. In this paper we use the HDFS data set to test our adapted LogDeep implementation and to compare it with published results of [8].

3.2 Audit Data set

This data set was provided by researchers of AIT and is also publicly available since 2020 [14]. The authors show in [15], how to set up a testbed and generate the Audit data set. They published four testbeds, where they log users accessing a Webmail platform and online store in a time period of six days. Within this time period one multistep attack and one complex vulnerability exploit were launched. The most difficult task related to data set creation is to correctly label log lines. For this data set the authors used time-based labels and line-based labels. Time-based labeling uses time stamps, which are parsed with each log line. An attack is labeled malicious, when the log line occurrence lies within the time frame of the attack stage. Problems with this approach arise, if normal log lines interleave with log lines of an attack. So the second approach called line-based labeling tries to overcome this shortcoming by labeling attack steps in an idle system. It turns out that most attacks generate ordered log-event sequences (e.g., Webshell upload) or repeating log lines (e.g., scans). Through manual review, line-based labels have proven to be more accurate for the Audit data set.

3.3 Feature Selection and Parsing

The Audit logs are collected by a Linux Audit daemon. Those logs are interesting, because they provide low-level syscall information and show accesses of files and paths on the host. Audit log lines include types such as SYSCALL-, CWD-, PATH- or PROCTITLE type. The type of each log line is specified at the beginning of the log line in the type field. Type SYSCALL indicates that a log line was triggered by a system call to the kernel. Type CWD does not occur in the data set provided by the AIT. Type PATH specifies all paths that a system call gets as arguments. Finally, Type PROCTITLE entry gives the complete command-line in hexadecimal notation. In this paper we concentrate only on the entries of type SYSCALL, because they contain all relevant information and the remaining type fields do not always occur. Each SYSCALL log line has several features. An example of an Audit log line can be seen in Fig. 2.

```
type=SYSCALL msg=audit(1583279999.201:38279754): arch=c000003e syscall=0
success=yes exit=1166 a0=10 a1=7f2f8668f048 a2=1f40 a3=5637e8a51ea0
items=0 ppid=22418 pid=17343 auid=4294967295 uid=33 gid=33 euid=33 suid=33
fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none) ses=4294967295 comm="apache2"
exe="/usr/sbin/apache2" key=(null)
```

Figure 2: Example of an Audit log line.

To make the features accessible from raw logs we use a parser. With log parsing we extract event templates from unstructured raw log data. Each log message consists of a constant event template and a variable part, where values can differ over time. The goal is to separate constant- from variable parts and form a log event. In [8] they use an effective methodology to extract so called log keys. Those log keys represent the constant part of a print statement in given log entries. For our approach we used a state of the art parser called Spell [7], which utilizes a Longest Common Subsequence (LCS) based method. The LCS method is based on the assumption that log lines mostly consist of constant parts and only a small part is coverd by variables. If two log lines are produced by the same log printing statement and only differ in the dynamic variable part, the LCS of the two log lines is very likely to be constant and thus an event template. In a log line all occurring words are so-called tokens separated by delimiters. These delimiters depend on the format of the log line. Tokens such as syscall or process identifier (pid) are used for the construction of the LCS [7]. In our approach log lines were grouped based on the pid feature first. This way, we get all log lines that belong to a single process. With the Spell parser, the grouped log lines were parsed based on the syscall feature. Thus, each syscall that occurs in the data set, corresponds to a log key. There are 268 unique log keys in total. The result of these steps are ordered log-event sequences that can be used as input for the LSTM. The LSTM divides these log-event sequences into so-called windows and processes them one after the other. Windows can be based on time or sessions. Time-based grouping is again divided into sliding time windows and fixed time windows. For sliding time windows, a duration is specified and pushed over all data with a static step size. For each step all log lines are grouped together that are between a start and end time of a time window. By moving the window further, an overlapping of log lines is possible. On the other hand for fixed windows, the step size is the same as window size. Log lines are always considered in exactly one window size and overlapping is not possible. Fixed time windows give a less fine-grained view on log lines but each log line is also only considered once. In contrast, session windows group log lines by session identifiers. This way, program flows can be represented very well, which may be processed at different times. It can also be used to distinguish between events that may take place in parallel. Unfortunately, not all types of log data have session identifiers [13]. For the processing of Audit log lines we use a combination of sliding and session window. This way we can use the advantages of both. As session identifier we use the pid feature. For HDFS log data "block_id" serves as session identifier, which marks a particular execution sequence and groups together particular log lines.

3.4 LogDeep Adaption

A detailed search of codebases such as Github and Gitlab revealed LogDeep [6] to be the most promising re-implementation, based on ratings and year of release. LogDeep combines the work of [8],[28] and [17] to a framework for AD utilizing log data and has already been adapted in other scientific works, such as [10]. Because of the scientific relevance, we focused on the DeepLog part from this implementation. As previously described, DeepLog uses LSTM as DL model. The main adjustable parameters in the LogDeep implementation are:

- L: Number of layers of LSTM.
- α: Number of memory units in one LSTM block.
- Window size: Length of window under consideration.
- Candidates: Number of candidates that were predicted with their respective probabilities.

After the parameters have been set, LogDeep reads the training log-event sequences. The log-event sequences must have at least the length of the specified window size. In principle, training of the DL model and AD, that uses the trained model, are separate processes. The reason for this is that after the training, the model is stored and can be transferred somewhere else in order to perform AD there as well. The Adam optimizer is used for the LSTM. This optimizer is often used for tasks where sparse gradients are to be expected. The advantages of this optimizer are increased computational performance and low memory requirements [12]. The LSTM delivers candidates with a certain probability which log key is expected next in the log-event sequences. Therefore, we use the cross entropy as loss function here, which quantifies the difference between probability distributions. After training, LogDeep can be used for AD. For this purpose, normal and anomalous test data are read in separately. The model calculates which log key comes next for each individual window and compares whether this matches the log keys in the test data. If the following log key is not contained in the proposed candidates, the whole log-event sequence is marked as anomaly.

3.4.1 *Padding.*

While analyzing the source code of LogDeep, we came across an interesting fact. If we take a closer look at the HDFS data set, we see that train and test normal log-event sequences consist of at least 10 consecutive log keys. However, in the test abnormal log-event sequences, also shorter sequences occur. This results in a problem for the window size, because the LSTM determines how far back it stores past information. In order to be able to analyze these short sequences with the model, so-called padding is applied. In this case, log-event sequences are filled with a log key that not occured in the data, in order to achieve desired window size. This log key must be unique and must not have occurred before, otherwise the data set would be corrupted. Based on this discovery, we conducted another experiment to show what effect padding has on LogDeep. Section 4.1 presents the related results. If padding is omitted, LogDeep cannot process and discards all test abnormal log-event sequences in the AD phase, which number of log keys is less than the window size. The size of the training and test normal data set remains the same, only the size of the test abnormal data set is reduced.

4 EVALUATION

The re-implementation of DeepLog called LogDeep [6] was used as baseline for all further experiments. In the first step the code base was analyzed. With the original code base only the HDFS data set can be processed. The code has been adapted to allow a user to choose if they want to process a HDFS or an Audit data set. In our implementation the parameters for the selected data set can be set and padding, which is described in Sect. 3.4.1, can be activated or deactivated. The following requirements for the current implementation apply: python >= 3.6 and pytorch >= 1.1.0.

4.1 LogDeep with HDFS data set

First, we verified if the evaluation metrics of LogDeep correspond to those published in [8]. For this purpose we set the window size to 10. The hidden size, which defines the number of hidden states, was set to 64. Further, the number of layers was set to 2, number of candidates to 9, batch size to 2,048, and number of epochs to 300. These values were also taken from [8]. As a basis for evaluation we used the freely available HDFS data set in parsed form and preselected splitting of train and test data [1]. This way our implementation can be compared with other scientific publications and errors caused by parsing can be omitted. The number of log keys is 29 and the exact breakdown of the data set can be taken from Tab. 1.

Train sequences	Test normal sequences	Test abnormal sequences
4,855	553,366	16,838

Table 1: Splitting of HDFS data set in train- and test sequences.

Results show that the re-implementation achieves approximately the same metrics as in [8]. The comparison is listed in Tab. 2.

	Accuracy	Precision	Recall	F1-Score
DeepLog [8]	-	95%	96%	96%
LogDeep [6]	99.76%	95.63%	96.35%	95.99%

Table 2: Comparison between DeepLog and LogDeep.

The effects of padding can be analyzed well using the HDFS data set. Since only the test abnormal sequences contains short log-event sequences smaller than the window size, padding is only applied there. By appending a unique log key, the log-event sequences are artificially extended in order to be processed with the LSTM. In principle, this results in a trade off. On the one hand, artificial lengthening gives better results for the metrics. By attaching a unique log key to the test abnormal sequences, the AD is very simple and it has the same effect as attaching an anomaly label. Therefore, the results are artificially improved due to padding, which is misleading. On the other hand, without padding, log-event sequences that are too short would simply be discarded and not fed into the AD process. The number of test abnormal sequences is reduced from 16,838 to 10,647. The small amount of test abnormal sequences compared to the test normal sequences indicates an imbalanced data set. The results of this experiment can be seen in Tab. 3 and Fig. 3.

	Accuracy	Precision	Recall	F1-Score
padding	99.66%	95.13%	95.81%	94.45%
nopadding	99.66%	93.07%	89.03%	91.00%

Table 3: Comparison between padding and nopadding for LogDeep with HDFS log-event sequences.

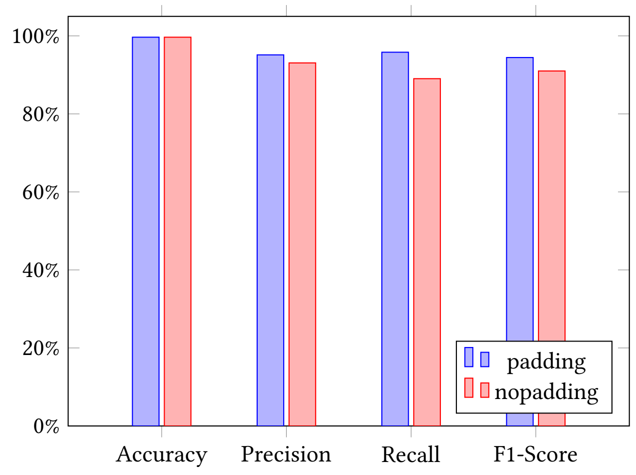

Figure 3: Comparison between padding and nopadding for LogDeep with HDFS log-event sequences.

4.2 LogDeep with Audit data set

The preprocessing- and parsing steps for the Audit [15] data set were described in Sect. 3. The baseline of the Audit data set consists of 18,428,737 log lines. In contrast to the HDFS data set, the common split of data sets in DL of 80% train data to 20% test data was used [5][2][20]. The exact separation into training, test normal and test abnormal sequences can be seen in Tab. 4. The LSTM was not fundamentally changed and we use padding in the first step.. We empirically assessed that the LSTM achieves best performance when the number of candidates are set to 6 and number of epochs are set to 20 for the Audit data set.

Log lines	Train sequences	Test normal sequences	Test abnormal sequences
~18 mil.	33,634	8,409	90

Table 4: Splitting of Audit data set in train- and test sequences.

The results for a LogDeep approach with ~18 million Audit log lines and padding can be taken from Tab. 5. When analyzing the Audit log-event sequences, we see that 39.56% of the test abnormal sequences also occur in the train sequences and 28.67% of the test abnormal sequences occur in the test normal sequences. However, we have extended the analysis to not only look at identical log-event sequences but also to check if test abnormal sequences occur at some arbitrary point in training or test normal sequences. An Audit log-event sequence is built up on individual log keys, e.g., syscalls. A practical example could be: If we log the behavior of a user, the sequence of syscalls which are executed is not clear from the start or can also occur in a different order. To continue the example, one can assume an attacker, who behaves like a normal user up to a certain point. Thus, the beginning of an Audit log-event sequence can look exactly the same in the benign as well as in the anomaly case. This makes it difficult to distinguish between benign and malicious log-event sequences and leads to poor Recall scores, which is the most important value for accurate AD. One recognizes a limitation here due to the recorded features of Audit logs. Another important point for the Audit data set is that it is an imbalanced data set. As

Tab. 4 shows we have only 90 malicious log-event sequences. The low number of test abnormal sequences compared to train and test normal sequences is the reason for the high Accuracy values.

If the Audit log-event sequences are not padded, the number of train sequences is reduced from 33634 to 32822, the number of test normal sequences from 8409 to 8169 and the number of test abnormal sequences from 90 to 44. The results for the reduced data set can be seen in Tab. 5 and Fig. 4.. It can be seen that some metrics for both data sets are degraded by nopadding. But the Recall seems to increase. The reason for this is that we have test abnormal sequences that are detected regardless of whether they are padded or not. Those sequences are very long sequences and stand out from the rest. Nopadding reduces the number of processable anomalies by half. This reduction increases the Recall despite the same number of correctly detected sequences. Nopadding is also not an option because too short log-event sequences are simply omitted. In our opinion the effect of padding for Audit data is also not as poor as for the HDFS data, because the same unique log key is appended to the train, test normal and test abnormal sequences.

	Accuracy	**Precision**	**Recall**	**F1-Score**
padding	99.04%	68.00%	18.88%	29.56%
nopadding	99.53%	58.07%	40.91%	48.00%

Table 5: Comparison between padding and nopadding for LogDeep for Audit log-event sequences.

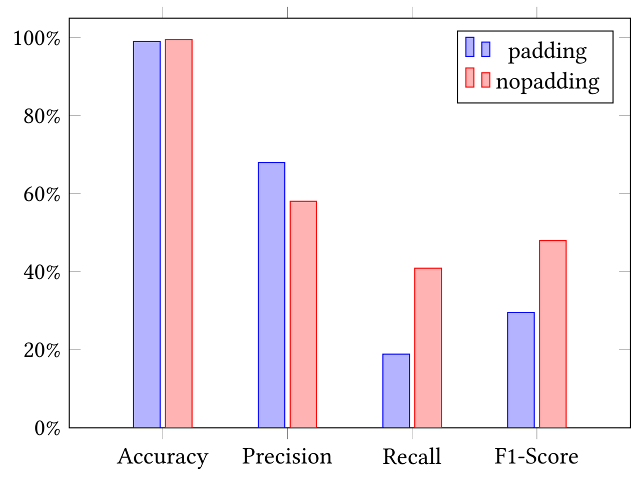

Figure 4: Comparison between padding and nopadding for LogDeep for Audit log-event sequences.

5 DISCUSSION

If we want to detect contextual anomalies we can use LSTM models in general and our LogDeep re-implementation as the results show. While validating already published scientific results for the HDFS data set, we noticed that a specific padding function was used. Padding results in an artificial lengthening of log-event sequences. That means if log-event sequences are not at least as long as the window size they will not be considered at all and discarded. Especially for the HDFS data set this has undesired effects. The train- and test normal sequences are sufficiently long log-event sequences. This is important for the applied window size of the LSTM model. In contrast to this, the test abnormal sequences are artificially lengthened with a unique log key. For AD it is very simple to detect these log-event sequences, because they do not occur during training and testing with normal sequences. This indicates better performance than can be achieved in real operation. After validating the results for the HDFS data set, we turned to the Audit data set. We have chosen the syscall feature as log key. Experiments have shown that due to the small number of distinct values for the syscall and the occurrence of identical log-event sequences in the train-, test normal-, and test abnormal sequences, a precise AD is not possible. Another important point for the Audit data set is that it is an imbalanced data set, like the HDFS data set. The Audit data set consists of many more benign log lines than malicious log lines. For the investigated Audit data set, the multiple occurrences of log lines in the train-, test normal-, and test abnormal sequences cannot be avoided, because attackers partly behave the same way as normal users. That means, up to a certain point the behavior looks identical to the logging system, because the same commands are executed. It would be necessary to improve the logging fundamentally and to collect more features. In this paper we decided not to simply discard the multiple occurrences of Audit log lines just to achieve better metrics. Simply deleting these multiple occurrences would falsify the existing Audit data set. We have also investigated the effects of padding for the Audit data set. In contrast to the HDFS data set, the train- , test normal-, and test abnormal Audit sequences are shortened log-event sequences with respect to the window size. When artificially lengthened with a unique log key, padding has the same effect on both data sets. With these limitations in mind, further scientific research challenges in this domain should be investigated:

- Further research would be needed to determine if LSTM is the best DL model to analyze log-event sequences. We are currently seeing a development of combining individual DL models that were previously treated separately and that by combining them, possibly better results can be achieved [24].
- In our approach we grouped individual log lines by their pid and used the syscall feature as log key. However, since Audit log lines have also other features, it would be of interest to know if AD improves when using other features or a variety of features as log keys.
- The handling of imbalanced data sets where many normal log lines face very few anomalous log lines would need further investigation.

6 CONCLUSION AND FUTURE WORK

In this paper we showed the necessary design steps, prerequisites to consider, and an approach for AD utilizing DL which can handle both HDFS and Audit data sets. Audit data sets were not used extensively in this area of scientific research in the past. Our implementation uses a LSTM and AD for log-event sequences. In the course of implementation, limitations came to light that should be considered in future design processes. It is imperative to have a basic understanding of how the log data to be analyzed are collected and structured. One of the biggest challenges for AD is to find the right DL method for the use case at hand and its suitability for a given data set.

We adapted a state of the art open source application called LogDeep and were able to replicate the results from paper [8] which uses a HDFS [26] data set. However, we found a limitation that occurs due to the padding function. Through this function it is possible to extend too short log-event sequences with respect to the window size, which is important for the application of LSTM. This has to be done in the test abnormal sequences set in such a way that AD is almost trivial for these artificially modified log line sequences. Consequently, the very good detection performance reported in paper [8] cannot be achieved without the padding function. Afterwards, we adapted LogDeep for an Audit data set. Unfortunately, we could not achieve nearly satisfactory values for the metrics of AD. The padding function described above did not improve the results. One of the reasons for the poor results is that the Audit data set is imbalanced. In this data set there is an overabundance of benign log-event sequences compared to a small number of malicious log-event sequences. Furthermore, a large number of log-event sequences were again found to be identical in the benign and malcious data set. This is again due to the labelling problem described above. Due to the limitations described in this paper, further scientific work is needed to achieve satisfactory AD.

ACKNOWLEDGMENTS

This work was partly funded by the European Defence Fund (EDF) project AInception under grant no. 101103385.

REFERENCES

[1] Anonymous. 2021. Loghub. https://zenodo.org/record/3227177.

[2] Salah Bouktif, Ali Fiaz, Ali Ouni, and Mohamed Adel Serhani. 2018. Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches. *Energies* 11, 7 (2018). https://doi.org/10.3390/en11071636

[3] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep Learning for Anomaly Detection: A Survey. http://arxiv.org/abs/1901.03407

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *Comput. Surveys* 41, 3 (2009), 1–58. https://doi.org/10.1145/1541880.1541882

[5] P. Dangeti. 2017. *Statistics for Machine Learning.* Packt Publishing.

[6] Donglee-Afar. 2020. LogDeep. https://github.com/donglee-afar/logdeep.

[7] Min Du and Feifei Li. 2016. Spell: Streaming Parsing of System Event Logs. In *IEEE 16th International Conference on Data Mining.* 859–864. https://doi.org/10.1109/ICDM.2016.0103

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 1285–1298. https://doi.org/10.1145/3133956.3134015

[9] Amir Farzad and T. Aaron Gulliver. 2020. Unsupervised log message anomaly detection. *ICT Express* 6, 3 (2020), 229–237. https://doi.org/10.1016/j.icte.2020.06.003

[10] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In *International Joint Conference on Neural Networks.* 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534113

[11] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In *IEEE 27th International Symposium on Software Reliability Engineering.* 207–218. https://doi.org/10.1109/ISSRE.2016.21

[12] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980

[13] Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. 2022. Deep Learning for Anomaly Detection in Log Data: A Survey. http://arxiv.org/abs/2207.03820

[14] Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner, and Andreas Rauber. 2020. AIT Log Data Set V1.1. https://zenodo.org/record/4264796.

[15] Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner, and Andreas Rauber. 2021. Have it Your Way: Generating Customized Log Datasets With a Model-Driven Simulation Testbed. *IEEE Transactions on Reliability* 70, 1 (2021), 402–415. https://doi.org/10.1109/TR.2020.3031317

[16] Hongyu Liu and Bo Lang. 2019. Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences* 9, 20 (2019), 4396. https://doi.org/10.3390/app9204396

[17] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence.* 4739–4745. https://doi.org/10.24963/ijcai.2019/658

[18] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. *CoRR* (2020). https://arxiv.org/abs/2008.09340

[19] Adam Oliner and Jon Stearley. 2007. What Supercomputers Say: A Study of Five System Logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* 575–584. https://doi.org/10.1109/DSN.2007.103

[20] Santwana Sagnika, Bhabani Shankar Prasad Mishra, and Saroj K. Meher. 2021. An attention-based CNN-LSTM model for subjectivity detection in opinion-mining. *Neural Computing and Applications* 33, 24 (2021), 17425–17438. https://doi.org/10.1007/s00521-021-06328-5

[21] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. 2007. Conditional Anomaly Detection. *IEEE Transactions on Knowledge and Data Engineering* 19, 5 (2007), 631–645. https://doi.org/10.1109/TKDE.2007.1009

[22] Miryam Elizabeth Villa-Pérez, Miguel Á Álvarez Carmona, Octavio Loyola-González, Miguel Angel Medina-Pérez, Juan Carlos Velazco-Rossell, and Kim-Kwang Raymond Choo. 2021. Semi-supervised anomaly detection algorithms: A comparative summary and future research directions. *Knowledge-Based Systems* 218 (2021). https://doi.org/10.1016/j.knosys.2021.106878

[23] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. 2019. Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access* 7 (2019), 41525–41550. https://doi.org/10.1109/ACCESS.2019.2895334

[24] Qiaozheng Wang, Xiuguo Zhang, Xuejie Wang, and Zhiying Cao. 2022. Log Sequence Anomaly Detection Method Based on Contrastive Adversarial Training and Dual Feature Extraction. *Entropy* 24, 1 (2022). https://doi.org/10.3390/e24010069

[25] Markus Wurzenberger, Florian Skopik, Giuseppe Settanni, and Roman Fiedler. 2018. AECID: A Self-learning Anomaly Detection Approach based on Light-weight Log Parser Models:. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy.* SCITEPRESS, 386–397. https://doi.org/10.5220/0006643003860397

[26] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles.* ACM Press, 117. https://doi.org/10.1145/1629575.1629587

[27] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. PLELog: Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In *IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings.* 230–231. https://doi.org/10.1109/ICSE-Companion52605.2021.00106

[28] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. 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.* 807–817. https://doi.org/10.1145/3338906.3338931

Received 11 January 2023; revised 24 March 2023; accepted 28 March 2023