Recompose Event Sequences vs. Predict Next Events: A Novel Anomaly Detection Approach for Discrete Event Logs

Lun-Pin Yuan
Pennsylvania State University
United States
lunpin@psu.edu

Peng Liu
Pennsylvania State University
United States
pxl20@psu.edu

Sencun Zhu
Pennsylvania State University
United States
sxz16@psu.edu

ABSTRACT

One of the most challenging problems in the field of intrusion detection is *anomaly detection for discrete event logs*. While most earlier work focused on applying unsupervised learning upon engineered features, most recent work has started to resolve this challenge by applying deep learning methodology to abstraction of discrete event entries. Inspired by natural language processing, LSTM-based anomaly detection models were proposed. They try to predict upcoming events, and raise an anomaly alert when a prediction fails to meet a certain criterion. However, such a *predict-next-event* methodology has a fundamental limitation: *event predictions may not be able to fully exploit the distinctive characteristics of sequences*. This limitation leads to high false positives (FPs). It is also critical to examine the structure of sequences and the bi-directional causality among individual events. To this end, we propose a new methodology: *Recomposing event sequences as anomaly detection*. We propose DabLog, a LSTM-based ***D**eep **A**utoencoder-**B**ased anomaly detection method for discrete event **Log**s*. The fundamental difference is that, rather than predicting upcoming events, our approach determines whether a sequence is normal or abnormal by analyzing (encoding) and reconstructing (decoding) the given sequence. Our evaluation results show that our new methodology can significantly reduce the numbers of FPs, hence achieving a higher F_1 score.

CCS CONCEPTS

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

KEYWORDS

Computer Security, Anomaly Detection, Machine Learning

ACM Reference Format:
Lun-Pin Yuan, Peng Liu, and Sencun Zhu. 2021. Recompose Event Sequences vs. Predict Next Events: A Novel Anomaly Detection Approach for Discrete Event Logs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Virtual Event, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3433210.3453098

ASIA CCS '21, June 7–11, 2021, Virtual Event, Hong Kong

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453098

1 INTRODUCTION

One of the most challenging problems in the field of intrusion detection is *anomaly detection for discrete event logs*. Researchers have been trying to resolve this challenge for two decades, and most work have focused on applying unsupervised learning upon engineered features from normal data, assuming unforeseen anomalies do not follow the learned normal patterns (e.g., [1, 18, 20, 21, 23–25, 27, 28, 31]). Recently, solving this challenge with deep learning has gained a substantial amount of traction in the security community (e.g., [4, 10–12, 40]), partially due to the unique advantages of deep learning in natural language processing. Researchers have applied related language-processing methodologies to anomaly detection for discrete event logs by treating discrete events as words and logs as sentences, as if linguistic causality exists in the security logs. The main benefit of this approach over machine learning upon engineered features is that detailed domain knowledge, complex feature extraction, and costly human interference are no longer required (e.g., [9, 22, 23, 35, 41]).

Inspired by natural language processing, Long Short Term Memory (LSTM) [17] based anomaly detection models (e.g., [4, 10–12, 40]) were proposed. These models try to predict upcoming log events, and they raise an anomaly alert when a prediction fails to meet a certain criterion. However, we found that the widely-adopted methodology *"using an LSTM-based model in predicting next events"* has a fundamental limitation: ***event predictions may not be able to fully exploit the distinctive characteristics of sequences***. To be specific, event-prediction methodology assumes the distribution of an event is affected only by the prior events before it (e.g., when a model sees an *open* file-operation, it can guess such an *open* operation is followed by *read* operations); however, the distribution can also be affected by later events (e.g., when a model sees a *read* operation, it should examine whether it has seen any *open* operation) or no events whatsoever (e.g., an event may have nothing to do with the other events). Therefore, an anomaly detection method should also look deeper into the sequential structure and the bi-directional causality among events. Because of this limitation, the widely adopted methodology could lead to numerous false positives (FPs).

The reason why the methodology could lead to FPs is illustrated in the following example. Consider a normal sequence of file operations [*open A*, *read A*, *read A*] and an upcoming event *open B*. By seeing just the first few operations, a predictor-based anomaly detection model may guess the upcoming event to be *read A*, because (1) it is one of the most frequent events that follow *open A* in the training dataset while *open B* is less frequent, and (2) the first few operations does not enclose prior knowledge which indicates *B* will be soon opened; consequently, the predictor-based model may

wrongly report the sequence as abnormal, although in reality it is also normal but less common. The fundamental issue is that, when little necessary knowledge is available in a sequence (regardless of sequence length), predictor-based anomaly detection always has to make bold guesses. As an FN (false negative) example, consider an abnormal sequence of file operations [*read A*, *read A*, *close A*] and an upcoming event *read A*. If the predictor-based model does not examine whether there is any *open A* before the upcoming *read A*, it may consider this sequence normal, hence a false negative.

To address the fundamental limitation of *not being able to fully exploit the distinctive characteristics of sequences*, we propose a different methodology: ***using an LSTM autoencoder to recompose sequences***. Compared to the existing methodology, the fundamental difference is that our LSTM autoencoder determines whether a sequence is normal or abnormal by analyzing (encoding) and reconstructing (decoding) the given sequence rather than predicting upcoming individual events. The intuition is that an anomaly detection method should see a sequence as an atomic instance, and it should examine the structure of the sequence as well as the bi-directional causality among the events. By doing so, our anomaly detection model can detect not only sequences that include unseen or rare events, but also structurally abnormal sequences. Note that, our model is more than a standard autoencoder which reconstructs input vectors. To work with discrete events, our solution is designed as an *embed-encode-decode-classify-critic* model.

In this work, we propose DabLog, a ***D**eep **A**utoencoder-**B**ased anomaly detection method for discrete event **Log**s*. DabLog aims to provide an anomaly detection function $\mathcal{AD} : \mathcal{S} \rightarrow$ {normal, abnormal}. DabLog consists of four major components (Figure 2): an embedding layer, a deep LSTM autoencoder, an event classifier, and an anomaly critic. Our evaluation results show that the new methodology can significantly reduce the number of FPs, while achieving a better F_1 score. Compared to our predictor-based baseline model, DabLog reports 1,790 less FPs but 1,982 more TPs (true positives) in our evaluation upon system logs with 101 distinct events, and DabLog reports 2,419 less FPs with trade-off 83 less TPs in our evaluation upon traffic logs with 706 distinct events. Specifically, we make the following contributions.

(1) Through in-depth FP and FN case studies, we discover a fundamental limitation of predictor-based models. We resolve this limitation by proposing a deep autoencoder-based anomaly detection method for discrete event logs.
(2) We evaluate DabLog upon two datasets and our results show that DabLog outperforms our re-implemented predictor-based baseline model in terms of F_1 score. DabLog achieves 97.18% and 80.25% F_1 scores in evaluation upon HDFS system logs (101 distinct events) and in evaluation upon UNSW-NB15 traffic logs (706 distinct events), respectively, while our baseline model achieves only 87.32% and 56.77%.
(3) To the best of our knowledge, we are the first to show that autoencoders can effectively serve the purpose of detecting *time-sensitive* anomalies in discrete event logs with more distinct events, while the common practice is to apply predictors to time-sensitive data with fewer distinct events and to apply autoencoders to time-insensitive data.

2 RELATED WORK

Most anomaly detection methods are *zero-positive* machine learning models that are trained by only normal (i.e., negative) data and then used in testing whether observation data is normal or abnormal, assuming unforeseen anomalies do not follow the learned normal patterns. For example, Kenaza et al. [20] integrated supports vector data description and clustering algorithms, and Liuq et al. [25] integrated K-prototype clustering and k-NN classification algorithms to detect anomalous data points, assuming anomalies are rare or accidental events. When prior domain knowledge is available for linking causal or dependency relations among subjects and objects and operations, graph-based anomaly detection methods (such as Elicit [27], Log2Vec [21], Oprea et al. [31]) could be powerful. When little prior domain knowledge is available, Principal Component Analysis (PCA) based anomaly detection methods (for example, Hu et al. [18] proposed an anomaly detection model for heterogeneous logs using singular value decomposition) could be powerful. Oppositions to zero-positive anomaly detection are semi-supervised or online learning anomaly detection, in which some anomalies will be available over time [8].

Autoencoder framework is another PCA approach that is widely used in anomaly detection. Briefly speaking, a typical autoencoder-based anomaly detection method learns how to reconstruct normal data, and it detects anomalies by checking whether the reconstruction error of a data point has exceeded a threshold. To detect anomalies, Zong et al. [43] proposed deep autoencoding Gaussian mixture models, Chiba et al. [6] proposed autoencoders with back propagation, Sakurada and Yairi [34] proposed autoencoders with nonlinear dimensionality reduction, Lu et al. proposed MC-AEN [26] which is an autoencoder which is constrained by embedding manifold learning, Nguyen et al. proposed GEE [30] which is a variational autoencoder with gradient-based anomaly explanation, Wang et al. proposed adVAE [37] which is a self-adversarial variational autoencoder with Gaussian anomaly prior assumption, Alam et al. proposed AutoPerf [1] which is an ensemble of autoencoders accompanied by K-mean clustering algorithm, Mirsky et al. proposed Kitsune [28] which is an ensemble of lightweight autoencoders, Liu et al. [23, 24] proposed an ensemble of autoencoders for multi-sourced heterogeneous logs, and Chalapathy et al. [5] and Zhou et al. [42] proposed robust autoencoders.

The above methods only work with ***time-insensitive*** data (i.e., data points are independent to each other). To work with ***time-sensitive*** data (i.e., dependencies exist among data points), researchers have leveraged Long Short-Term Memory (LSTM) [17] in building anomaly detection models. LSTM has been widely used in learning sequences, and LSTM-based deep learning has been widely used to extract patterns from massive data. Since most cyber operations are sequential (e.g., as in timestamped audit logs), LSTM-based deep learning has great potential in serving anomaly detection applications. Inspired by natural language processing, Deeplog [10], Brown et al. [4], DReAM [12], HAbAD [11], and nLSALog [40] were proposed to build LSTM-based multi-class classifier in order to predict future log entries. We summarize these LSTM-based methods in the next section; for other anomaly detection methods, recent surveys and comparisons can be found in [2, 13].

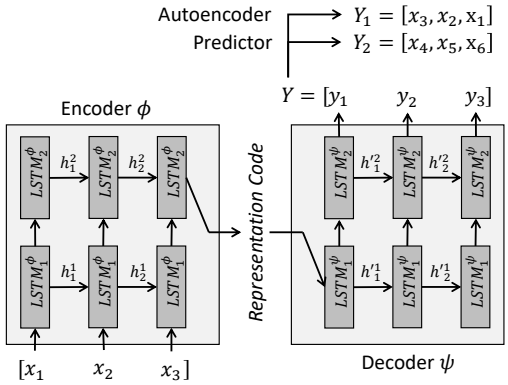


Figure 1: Deep LSTM Encoder-Decoder Network

3 BACKGROUND KNOWLEDGE

To understand our approach, some background knowledge on ***Deep LSTM Encoder-Decoder Network*** is essential. Cho et al. [7] proposed an LSTM encoder-decoder network for statistical machine translation. Both encoder and decoder are recurrent networks. An encoder ϕ takes a variable-length input sequence $X = [x_1, x_2, x_3, \ldots, x_T]$ of length T and generates a brief fixed-length ***representation code*** (also commonly referred to as the *representation* or the *code*) of X, and the encoding operation is denoted as code $= \phi(X)$). A decoder ψ then takes the representation code and generates a variable-length target sequence $Y = [y_1, y_2, y_3, \ldots, y_{\mathcal{T}}]$ of length $\mathcal{T}$, and the decoding operation is denoted as $Y = \psi(\text{code}) = (\psi \circ \phi)(X)$. Depending on the application, X and Y may have different lengths. Srivastava et al. [36] summarized three types of LSTM encoder-decoder networks for unsupervised learning models.

(1) ***Autoencoder***: The goal of an autoencoder is to enclose into the representation code all needed to *reconstruct the same sequence*. An autoencoder takes an input sequence $X = [x_1, x_2, x_3, \ldots, x_T]$ and tries to reconstruct the target sequence $\hat{Y}_1 = [x_T, x_{T-1}, x_{T-2}, \ldots, x_1]$. Note that the target sequence is in the reverse order, as if the encoder recurrently encodes (pushes) x_t into the representation code, whereas the decoder recurrently decodes (pops) x_t from the code.
(2) ***Predictor***: The goal of a predictor is to *predict future sequence* based on what it has observed. The representation code plays the role of an internal hidden state. A predictor takes the input sequence $X = [x_1, x_2, x_3, \ldots, x_T]$ and tries to predict the target sequence $\hat{Y}_2 = [x_{T+1}, x_{T+2}, x_{T+3}, \ldots, x_{T+\mathcal{T}}]$. If $\mathcal{T} = 1$, then it is a single-event predictor.
(3) ***Composite***: Merging the above two models, a composite model tries to reconstruct-predict $\hat{Y}_3 = [\hat{Y}_1, \hat{Y}_2]$.

Each LSTM encoder-decoder network listed above can be ***conditional*** or ***unconditional***, depending on whether the true $\hat{y}_\tau$ (*condition*) is provided to the decoder (as an additional input) when the decoder tries to decode $y_{\tau+1}$. In an autoencoder, $\hat{y}_\tau = x_{T-\tau+1}$, whereas in a predictor $\hat{y}_\tau = x_{T+\tau}$.

The time-sensitive anomaly detection models for discrete event logs, mentioned in Section 2, are predictors. These predictors typically consider an upcoming event $x_{T+\tau}$ normal if the probability $\Pr(x_{T+\tau}|x_1, x_2, \ldots, x_{T+\tau-1})$ is within a threshold (or alternatively $x_{T+\tau}$ is within the top-N prediction); otherwise abnormal. Specifically, DeepLog [10] leverages a two-layer LSTM network that works on one-hot representation of log entries. Brown et al. [4] leverages bidirectional LSTM, word embedding, and five attention mechanisms. Both HAbAD [11] and DReAM [12] build an embed-encoder-attention-decoder framework. nLSALog [40] leverages n-layer stacked LSTM, embedding layer, and self-attention mechanism. Some of the above models further incorporate an *embedding layer* (here embedding is a learned representation of log entries) in their encoders in order to include correlation among log entries, and some incorporate an *attention layer* (here attention is an aggregated state of hidden states from each time-step or each neuron) in their decoders in order to improve prediction accuracy.

Predictors and autocoders have been extensively studied for time-sensitive anomaly detection, and for time-insensitive anomaly detection, respectively. However, to our best knowledge, whether autoencoders can serve time-sensitive anomaly detection have not yet been investigated until this work. Conceptually, an LSTM autoencoder is essentially trying to learn the identity function of the input data distribution. Such identity function will definitely fail to fit every input data because, at high-level, there are only a fixed number of hidden units at each layer (in both encoder and decoder) and thus very unlikely they can learn everything needed for reconstruction. Moreover, hidden states (e.g., h_j^i and $h_j'^i$ in Figure 1) and representation codes are too small to enclose detailed information of the input data. Based on these constraints, autoencoders are forced to learn more meaningful concepts and relationships inside the input data. Trained with only normal input, autoencoders can be used in detecting anomalies in case of poor reconstruction.

4 MOTIVATION

We define an anomaly detection function for discrete events as $\mathcal{AD} : \mathcal{S} \rightarrow \{\text{normal}, \text{abnormal}\}$, where a sequence of events $S = [e_t | 1 \leq t \leq T] \in \mathcal{S}$ is essentially a set of relevant events (e.g., events of the same subject) sorted by timestamps (e.g., from past to present). Each discrete event e_t is represented by a distinct event key $k_i \in \mathcal{K}$, which is a string template. Distinct event keys are referred to as *logkey* in DeepLog [10], *log template* in nLSALog [40], and *discrete keys* in Du et al. [8]'s work.

Among the aforementioned related work, we find DeepLog [10] and nLSALog [40] representative of predictor-based anomaly detection methods. They are similar predictors that predict only single upcoming event e_{T+1} for each sliding-window subsequence $s_T = [e_t | \max(1, T-9) \leq t \leq T]$, whose window size $|s_T|$ is at most ten (we refer this configuration to as seqlen = 10). They consider S anomalous if any prediction $e_{T+1} \in S$ is not an instance of event key $k_j \in \mathcal{K}$ in its top-9 predictions out of 28 event keys, or equivalently top-32% predictions. Both methods were evaluated upon the same HDFS dataset [38, 39] and seemed promising based on accuracy $= (TP + TN)/(TP + FN + FP + TN)$.

Single-event prediction, however, is not an ideal solution for sequence-based anomaly detection $\mathcal{AD} : \mathcal{S} \rightarrow \{\text{normal}, \text{abnormal}\}$. Typical anomaly detection methods are based on the variance of an instance, or equivalently the error from particular expectation of an instance. In our context, the instances are the sequences $S \in \mathcal{S}$,

and hence intuitively we should consider an individual sequence S as an atomic instance. Yet, by examining whether an individual event $e_{T+1} \in S$ is in top-N expectation based on prior events, single-event prediction obviously considers the individual event e_{T+1} as an atomic instance. As such, it seems to us that single-event prediction is more like anomalous event detection, or somewhat rare event detection considering their configuration setup. The problem is twofold. On one hand, a rare event does not necessarily make the event itself or the sequence abnormal, and hence wrongly reporting rare events as abnormal will cause more FPs (a case study is provided below in the following subsection). On the other hand, the absence of abnormal events does not necessarily makes the sequence normal. That is, a sequence without abnormal events can still be structurally abnormal; therefore, not checking sequential structure may cause FNs (a case study is provided in Section 6). Based on the above observations, it is important to examine the structure of a sequence (and even to reconstruct it) as well as its bi-directional causality.

The anomaly criterion *"top-9 predictions out of 28 event keys"* with the HDFS dataset [38, 39] is also problematic: top-9 is too high and $|\mathcal{K}| = 28$ is too small. The reason is twofold. On one hand, since the top-9 most frequent event keys dominate 98.66% of the entire dataset, a trivial model that always blindly guess top-9 frequent keys can already achieve 85.58% accuracy and 14.33% FP rate, and yet we have no clue about *why a sequence is abnormal* besides event frequency. Apparently, we need a much more precise criterion, so that it is easier to tell *"what are normal"* in order to figure out *"why anomalies are abnormal"*. A more reasonable criterion could be top-3 or equivalently top-10% (top-3 event keys dominate 42.37% of the dataset). On the other hand, if we look at the distinct sequential patterns under configuration seqlen = 10, we have in total 28,961 patterns, among which 13,056 are always normal and 11,099 are always abnormal. The number of patterns is so small that any anomaly function, that merely learns these 13,056 normal patterns and reports the others as anomalies, can get reasonable results. This is the reason why prior work can be trained by only incredibly few data of size just 4,855 out of 575 thousand sessions. However, in practice, the number of distinct event keys can exceed a hundred, and the number of patterns can become unlearnable due to the scale. One may argue that the number of keys can be reduced by abstracting and aggregating multiple keys, but we argue that by doing so one may no longer know what exactly happened due to lack of critical fine-grained information.

4.1 Motivating Example: FP Case Study

We are particularly interested in how predictor-based approach can be applied to scenarios where finer grained prediction is required and more event keys are involved. We re-implement a predictor model (referred to as the *Baseline* model) which is similar to DeepLog [10, 38] and nLSALog [40], except it checks top-10% keys. We also re-engineered the event keys from the same HDFS dataset [39], so that we have $|\mathcal{K}'| = 101$. With $\mathcal{K}'$, we have in total 256,574 patterns, among which 220,912 are always normal and 35,662 are always abnormal. Trained by 200,000 sessions, Baseline can only achieve roughly 80% F_1 score.

We use the following FP case (session ID: -3547984959929874282) to motivate our autoencoder-based anomaly detection. The events are listed in Table 1. This session has in total 25 events, and the Baseline model reports the fifth event e_5 as an abnormal event. Baseline reports $e_5 = k_3$ as abnormal, because k_3 is not within the top-10% predictions for e_5. Top-10% predictions include the variants of k_4 = *"addStoredBlock: blockMap updated ..."*, k_5 = *"block terminating"*, and k_6 = *"Received block of size 60-70 MB from 10.251.∗"*. We can tell that k_3 and k_6 are variants of *"Received block of size ∗ from ∗"*. In fact, the corresponding embedded vectors $\mathcal{E}(k_3)$ and $\mathcal{E}(k_6)$ are close to each other in the hyper-dimensional embedding universe $\mathcal{U}$, meaning that their abstract concepts are similar in Baseline's point of view. However, the fact that k_3 is not in top-10% but at top-63% causes this FP.

The fundamental problem is that, without the pre-knowledge of the block-size information for this particular session, Baseline would rather guess *"60-70 MB"* as the block size by seeing just $s_4 = [k_1, k_2, k_2, k_2]$, since Baseline has learned through the training data that k_6 is a very frequent key (dominating 10.77% of the entire dataset), while k_3 is actually an extremely rare key (only dominating 0.05%). In other words, there are much more blocks of size *"60-70 MB"* than blocks of size *"20-30 MB"*. This problem is similar to the *cold-start* problem in a recommendation system, where, not knowing personal preference, a recommendation system often recommends new users with most popular products among the others. Similarly, not knowing information about this particular session, Baseline can only make bold guesses based on characteristics of the other sessions, and hence it produces this FP case. Once Baseline knows the size from e_5, it can then correctly predict events e_6, e_7, and e_8.

In contrast, an LSTM autoencoder-based anomaly-detection model can resolve this issue, by first analyzing (encoding) the sequence and then reconstructing (decoding) the sequence, as if the sequence is an atomic instance. By analyzing $s_{10} = [e_1, e_2, e_3, \ldots, e_{10}]$, an autoencoder model already knows that the transmission is of size *"20-30 MB"* and not *"60-70 MB"*, even though k_3 is an extremely rare event. Our autoencoder-based model not only can correctly reconstruct s_{10} with k_3 in e_5's top-10% reconstructions, but also can correctly reconstruct other subsequences from s_{11} to s_{15} that also involve e_5. As a result, our autoencoder model does not falsely report this session as abnormal. Furthermore, with configuration seqlen = 10 and top-10% criterion, our autoencoder model reported 3,187 less FPs and 2,145 more TPs than our Baseline model.

5 OUR DABLOG APPROACH

We propose DabLog, a ***D**eep **A**utoencoder-**B**ased anomaly detection method for discrete event **Log**s*. DabLog is an unsupervised and offline machine-learning model. The fundamental differences between DabLog and the aforementioned predictor-based related work is that, DabLog determines whether S is abnormal by reconstructing S rather than predicting (or sometime guessing) upcoming individual events. The intuition is that, to avoid guessing, an anomaly detection method should see a sequence as an atomic instance, and it should examine the structure of the sequence as well as the bi-directional causality among the events. In the event of poor reconstruction, DabLog can detect *not only sequences that include unseen or rare events, but also structurally abnormal sequences.*

Table 1: Example Sequential Discrete Events

		Event Key
e_0		<begin of sequence>
e_1	k_1	NameSystem.allocatedBlock /usr/root/...
e_2	k_2	Receiving block within the localhost
e_3	k_2	Receiving block within the localhost
e_4	k_2	Receiving block within the localhost
e_5*	k_3	Received block of size 20-30 MB from 10.250.*
e_6		blockMap updated: 10.251.* added of size 20-30 MB
e_7		blockMap updated: 10.251.* added of size 20-30 MB
e_8		blockMap updated: 10.250.* added of size 20-30 MB
e_9		PacketResponder 1 for block terminating
e_{10}		Received block of size 20-30 MB from 10.251.*

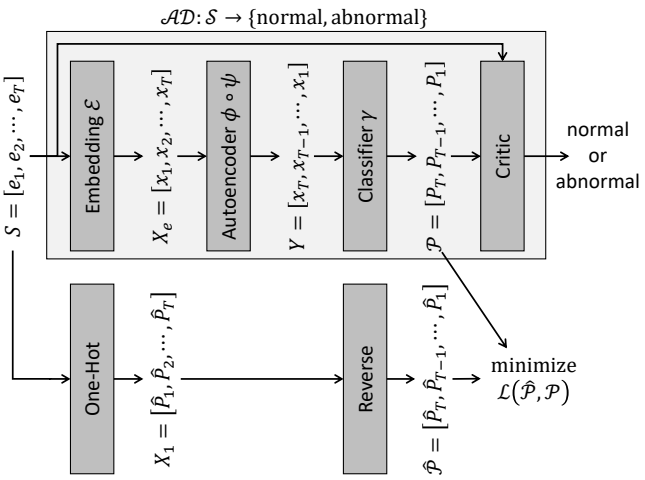


Figure 2: DabLog Anomaly Detection Model

DabLog focuses on ***discrete events***, which are essentially discrete-log representation derived from discrete log entries. Each log event e_t is represented as a ***discrete event key*** k_i (which is an abstraction string), and the key set is $\mathcal{K} = \{k_i | 1 \leq i \leq V\}$, where V is the number of unique discrete events (vocabulary size). Much work [9, 22, 35, 41] has been done for automatic discovery of unique discrete keys from security logs.

We make a ***Time-Sensitive Distribution Assumption:*** we assume that the value of the time-sensitive distribution $\mathcal{D}$ of an event e_t at time t may depend on both the past and future events; that is, one can expect both past and future causal events e_i and e_j when observing an event e_t, where $i < t < j$. For example, if e_t is *"deleting a remote object"*, then one can expect there is a past event e_i like *"ask to delete a remote object"* and a future event e_j like either *"deleted an remote object"* or *"deletion error"* (if such audit logs are available). We define the probability function of e_t for t in $i \leq t \leq j$ by $\Pr(e_t | e_i, e_{i+1}, \ldots, e_j)$; this assumption is not applicable to predictors, whose probability mass functions are typically defined by only the past, that is $\Pr(x_t | x_{t-1}, \ldots, x_1)$.

In summary, DabLog aims to provide an anomaly detection function $\mathcal{AD} : \mathcal{S} \rightarrow$ {normal, abnormal}. DabLog consists of four major components (Figure 2): an embedding layer, a deep LSTM autoencoder, an event classifier, and an anomaly critic. The workflow is stated as follows. Given a sequence $S \in \mathcal{S}$, the embedding layer $\mathcal{E}$ embeds S into an embedded distribution X_e, the autoencoder then analyzes (encodes) X_e and reconstructs (decodes) the categorical logit distribution Y, the event classifier then transforms Y into categorical probability distribution $\mathcal{P}$, and lastly the critic compares $\mathcal{P}$ with S and reports whether S is normal or abnormal.

5.1 Embedding Layer

Since our anomaly detection takes a sequence of discrete events $S = [e_t | 1 \leq t \leq T]$ as input, we need to embed discrete events $e_t \in \mathcal{K}$ into a particular model-recognizable vector, where $\mathcal{K} = \{k_i | 1 \leq k \leq V\}$ is the set of discrete event keys of vocabulary size $V = |\mathcal{K}|$. We denote an embedding function as $\mathcal{E} : \mathcal{S} \rightarrow \mathcal{X}$ and the procedure as $X_e = \mathcal{E}(S)$, where $X_e = [x_t | 1 \leq t \leq T] \in \mathcal{X}$ is an embedded distribution of S, and x_t is the embedded vector of e_t. There are three common embedding options adopted by prior work: (1) embedding with one-hot representation, (2) embedding using pre-trained natural linguistic packages, and (3) embedding by training an additional embedding layer along with the other layers.

We adopt the last option, because the other two options have major drawbacks. On one hand, one-hot representation, in which $x_t = [v_i | 1 \leq i \leq V]$ where $v_i \in \{0, 1\}$ and $\sum_i v_i = 1$, not only lacks the ability to embed the semantic or correlation features among keys, but also causes the model to suffer from dimension explosion when V is large (dimension explosion causes run-time inefficiency in machine learning). As a consequence, leveraging one-hot representation, DeepLog [10] does not work well on datasets with more keys (even the HDFS dataset), even though its accuracy has been improved by stacking two LSTM layers. On the other hand, while directly using pre-trained natural linguistic packages (e.g., Word2Vec and GloVe) seems convenient, it may not work well on security audit logs that lack natural linguistic properties [23]. The reasons include that (1) a JSON-formatted or CSV-formatted log may not demonstrate syntactic structure, (2) duplicate or redundant attributes may introduce unwanted noise, and (3) arbitrary abbreviated strings may not have a match in the existing packages.

Although the last option *training an additional embedding layer* is slower, the embedding function $\mathcal{E}$ can be well customized for the specific log dataset. That is, rather than no correlation (with one-hot representation) or syntactic correlation (using linguistic packages), the underlying correlation between discrete events $k_i \in \mathcal{K}$ (for example, k_3 and k_6 in Table 1) can be found by $\mathcal{E}$.

In our approach, an embedding layer is instantiated by V and the size of output dimension δ, and then it holds a random matrix that maps $e_t = k_i$ to x_t, where x_t is an embedded vector of size δ. This matrix is then trained by back propagation during its training phase along with the time-sensitive encoder-decoder network. In addition to event keys, we incorporate three special padding keys *begin-of-sequence*, *end-of-sequence* and *unknown* in our embedding layer. On one hand, the keys *begin-of-sequence* and *end-of-sequence* provide additional sequential characteristics to LSTM models, and we noticed a slight improvement for both autoencoders and predictors in detection results. On the other hand, the *unknown* key is used for improving computational performance. The problem of not using *unknown* key is that, the embedding function in prior work initiates

untrained embedding vectors for all unknown events, and similarly the event classifier also initiates unused logit dimensions for unknown events. It is inefficient to train such a machine-learning model when unknown events unnecessarily use much resource.

5.2 Deep LSTM Autoencoder

Deep autoencoders have been used in time-insensitive anomaly detection. Conceptually, an autoencoder learns the identity function of the normal data and reconstructs normal data distribution; hence the input data leading to poor reconstruction is potentially abnormal. Since we are tackling time-sensitive discrete events instead of engineered features, our autoencoder is different from typical ones that reconstruct the input features. Rather, it tries to reconstruct the logit distribution of categorical events.

A typical autoencoder is trained by minimizing the function: $\phi, \psi = \arg\min_{\phi,\psi} \|X - (\psi \circ \phi)(X)\|$, where ϕ is an encoder, ψ is a decoder, X is the input distribution, and $\psi \circ \phi(X)$ is the target (reconstructed) distribution. To tackle time-sensitive discrete events, our autoencoder (Figure 1) is trained by minimizing the function:

$$\begin{aligned}\phi, \psi &= \arg\min_{\phi,\psi} \|X - Y\|^2 \\ &= \arg\min_{\phi,\psi} \|\text{rev}(X_e) - (\psi \circ \phi)(X_e)\|^2 \\ &= \arg\min_{\phi,\psi} \|(\text{rev} \circ \mathcal{E})(S) - (\psi \circ \phi \circ \mathcal{E})(S)\|^2\end{aligned}$$

where ϕ is a deep encoder, ψ is a deep decoder, and rev is a function that reverses a distribution matrix. The encoder ϕ maps an $\mathcal{E}$-embedded matrix $X_e = \mathcal{E}(S)$ into a representation code $= \phi(X_e)$, whereas the decoder ψ maps the code into a target distribution matrix $Y = \psi(\text{code})$. Hence, the reconstructed distribution through the embed-encode-decode procedure is denoted as $Y = (\psi \circ \phi \circ \mathcal{E})(S)$. The function rev is involved because Y is in the reverse order from X_e due to LSTM's hidden state h_t, which is explained below.

We build our encoder ϕ and decoder ψ by stacking vanilla Long Short-Term Memory (LSTM) [17] (variants are applicable as well [15, 19]). The advantage of using an recurrent LSTM network over traditional recurrent neural networks is that an LSTM unit calculates a hidden state that conceptually remembers past activities as well as long-term dependencies [3]. For presentation purpose, we denote the computation of hidden state h_t by

$$h_t = \text{LSTM}\ (x_t, h_{t-1})$$

That is, at each time-step t, an LSTM unit takes two inputs x_t (the current data point) and h_{t-1} (previous hidden state), and it generates an output h_t (current hidden state). Multiple LSTM layers each calculates its own hidden state, as illustrated in Figure 1. The representation code $= \phi(\mathcal{E}(S))$ is essentially the transferable hidden state h_T at the last time-step T, and h_T is calculated by applying LSTM function iteratively from $t = 1$ up until the last time-step $t = T$. We can conceptually think of this procedure as *"pushing x_t into a state stack h_T"*; hence, the conceptual procedure for decoder is *"popping x_t out from a state stack h_T"*. Therefore, the distribution X_e and Y are in reverse order.

Our deep LSTM autoencoders are similar to traditional autoencoders that consist of deep encoders and deep decoders, except that our encoder ϕ and decoder ψ are implemented with stacked LSTMs (of at least two layers). The number of hidden units decreases (e.g., by half) layer-by-layer in ϕ, and increases (e.g., doubled) layer-by-layer in ψ. Some research work [14, 16, 32] have addressed the main benefit of stacking multiple LSTM layers over using a single layer: stacking hidden states potentially allows hidden states at each layer to reflect information at different timescale, and the final layer can gain benefits from learning or combining representations given by prior layers (hence better results).

Our autoencoder is *unconditional*, meaning that we do not provide a condition $\hat{y}_\tau = e_{T-\tau+1}$ to the decoder ψ when it is decoding $y_{\tau+1}$ for any y_τ in $Y = [y_\tau | 1 \le \tau \le T]$. This decision is made differently from some predictor-based anomaly detection methods [4, 10–12, 40] that either provide e_{T+k-1} to ψ when decoding e_{T+k} or predict only e_{T+1} for any input sequence $S = [e_t | 1 \le t \le T]$. Srivastava et al. [36] have shown that, although conditional decoders could provide slightly better results in predictors, unconditional decoders are more suitable for autoencoders. There are two reasons. First, autoencoders have only one expected output from any input sequence, which is the reconstruction of the input, whereas predictors could have multiple expected outputs (say S_1 and S_2 have the same prefix of length T but different suffices). While the condition acts as a hint about which suffix should be decoded in predictors, providing conditions serves no additional purpose in autoencoders. Second, usually there is strong short-term dependency among adjacent events, and hence it is not ideal to provide a condition that may cause the model to easily pick up short-term dependencies but omit long-term dependencies.

5.3 Event Classifier and Anomaly Critic

Since our goal is to provide an anomaly detection method $\mathcal{AD} : \mathcal{S} \rightarrow \{\text{normal}, \text{abnormal}\}$, simply combining an embedding layer and a deep LSTM autoencoder will not accomplish our goal. Similar to prior predictor-based anomaly detection methods [4, 10, 12, 40], right after our deep autoencoder, we add an additional single-layer fully connected feed-forward network γ, which is activated by a *softmax* function. The last layer γ acts as a multi-class classifier that takes input Y (which is the reconstructed distribution from ψ) and generates a probabilistic matrix $\mathcal{P} = [P_\tau | 1 \le \tau \le T]$, where $P_\tau = [p_i | 1 \le i \le V]$ and p_i can be interpreted as the likelihood of the discrete event $e_t = e_{T-\tau+1}$ being an instance of discrete event key k_i (that is, γ is an event classifier). We explain why we need γ in the following paragraph. In order to train γ, one-hot representation of S is provided as true probabilistic matrix, denoted as $X_1 = \text{onehot}(S) = [\hat{P}_t | 1 \le t \le T]$, where $\hat{P}_t = [\hat{p}_i | 1 \le i \le V]$ and $\hat{p}_i \in [0, 1]$ and $\sum_i \hat{p}_i = 1$. Similar to the embedding function $\mathcal{E}$, we also include three additional special padding keys *begin-of-sequence*, *end-of-sequence*, and *unknown* in the onehot function. Note that X_1 and $\mathcal{P}$ are in reverse order, so $\hat{\mathcal{P}} = \text{rev}(X_1)$. In our design, the multi-class classifier γ is trained by minimizing the

categorical cross-entropy loss function:

$$\mathcal{L}(\hat{\mathcal{P}}, \mathcal{P}) = \sum_{\tau}^{T} L(x_{T-\tau+1}, P_\tau), \text{ where}$$

$$L(x_t, P_\tau) = -\sum_{i}^{V} \hat{p}_i \times \log(p_i)$$

In summary, the overall embedder-encoder-decoder-classifier network tries to minimize the function:

$$\mathcal{E}, \phi, \psi, \gamma = \underset{\mathcal{E},\phi,\psi,\gamma}{\arg\min} \|(\text{rev} \circ \text{onehot})(S) - (\gamma \circ \psi \circ \phi \circ \mathcal{E})(S)\|$$

Unlike typical time-insensitive autoencoder-based anomaly detection methods, we do not directly use scalar reconstruction errors (e.g., root-mean-square error) as anomaly scores. The reason is that our problem—*identifying time-sensitive anomaly by examining discrete events*—is more like a language processing problem. We can view sequences S as sentences and events e_t as words, and we care more about wording (e.g., *"which words e_t better fit in the current sentence S"*) rather than embedding (e.g., *"which vector y_τ better vectorize e_t in the current sentence S"*). As such, we need the event classifier γ as well as certain reasonable *"wording options"* that help us with finding *"fitting words"* and *"unfitting words"*, or equivalently, normal events and abnormal events in S.

Rank-based criterion and threshold-based criterion are two common *"wording options"* adopted by previous predictor-based anomaly detection methods. Say a discrete event e_t is an instance of $\hat{k}_i$, a rank-based criterion will consider e_t anomalous if p_i is not in top-N prediction (e.g., $N = V/10$) in P_τ, and a threshold-based criterion will consider e_t anomalous if $p_i \in P_\tau$ is under a particular threshold θ_P. In other words, discrete keys $\{k_j | \forall j \text{ s.t. } p_j \in \arg\text{top}_N(P_\tau)\}$ and $\{k_j | \forall j \text{ s.t. } p_j > \theta_P\}$ are normal discrete keys. Our autoencoder-based anomaly detection adopts both threshold-based and rank-based criteria, but for presentation purpose we demonstrate the rank-based criterion (Figure 3) in this paper in order to compare our work with prior predictor-based methods [10, 40]. However, both rank-based and threshold-based criteria have the same major drawback: they brutally divide P_τ while omitting the correlation between the true $\hat{k}_i$ and the supposedly normal keys; hence, besides the aforementioned two criteria, our anomaly detection has an option of a novel criterion, which is discussed in Section 7.2.

Similar to prior anomaly detection work [10, 40] that conducted experiments on the same dataset [38, 39], in which anomaly labels (e.g., normal or abnormal) are given at the sequence level, our anomaly detection model gives labels to sequences. We say a sequence $S = [e_t | 1 \le t \le T]$ is abnormal if any $e_t \in S$ is abnormal. With aforementioned criteria, our anomaly detection method $\mathcal{AD} : \mathcal{S} \rightarrow \{\text{normal}, \text{abnormal}\}$ is complete.

6 EVALUATION

We motivate our evaluation with three questions: (1) how better is DabLog in comparison with a predictor-based baseline model, (2) how does *having more keys* impact the detection results, and (3) how does the fundamental difference make DabLog more advantageous. To answer these questions, we evaluate DabLog with two datasets:

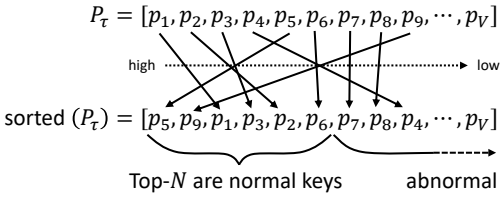


Figure 3: An example of the rank-based criterion

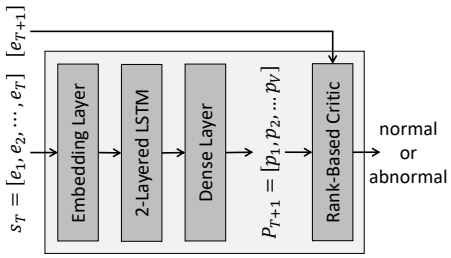


Figure 4: Predictor-Based Baseline Model

UNSW-NB15 [29] traffic logs and HDFS console logs [39]. This section, however, mainly focuses on the UNSW-NB15 dataset, as its context is more comprehensive and security related than the HDFS dataset. For the HDFS dataset, we will only briefly summarize our results due to the page limit.

6.1 DabLog and Baseline Implementation

We implement both Baseline and DabLog models with 3K lines of Python 3.7.4 scripts, and we leverage deep-learning utilities from Tensorflow 2.0.0. To train DabLog and Baseline, Adam optimizer is used with accuracy metric in minimizing categorical cross-entropy. We use the same sequence-length configuration seqlen = 10.

Baseline Model Implementation: Section 5.1 elaborated our design choice for adopting an embedding layer instead of one-hot representation. It will be unfair to compare our work with the vanilla DeepLog [10] due to its use of one-hot representation. Therefore, we slightly revise DeepLog and refer to it as the *Baseline* model. The performance difference between one-hot representation and embedding layer has been addressed in [4, 11, 12, 40]. The other parameters, including the numbers of layers and the numbers of hidden units, are the same. Similar to DeepLog and nLSALog [40], *Baseline* (illustrated in Figure 4) has a two-layer LSTM network, a multi-class classifier, and a rank-based critic, except that unlike DeepLog it does not learn one-hot representation, and unlike nLSALog it does not include a self-attention layer (note that, nLSALog's source code was not available when writing this paper). We leverage *tensorflow.keras.layers.Embedding* as the embedding layer, and we include three additional special padding keys *begin-of-sequence*, *end-of-sequence*, and *unknown* in learning sequences (they slightly improve the detection performance). The two-layered LSTM network is implemented by stacking two *tensorflow.keras.layers.LSTM* layers activated by ReLU. Each LSTM layer is configured to have 64

hidden units. The event classifier layer is implemented with *tensorflow.keras.layers.Dense*, which is activated by the *softmax* function. In event classifier, we include the three additional padding keys.

DabLog Model Implementation: Our encoder and decoder are implemented by stacking two *tensorflow.keras.layers.LSTM* layers, and each is activated by ReLU. The encoder is configured to have 64 and 32 hidden units for its 1st and the 2nd layer respectively, and the decoder is configured to have 32 and 64 hidden units for its 1st and the 2nd layer respectively, as we follow the common practice that the representation code is a downgraded abstraction. The embedding layer and the event classifier are implemented in the same way as the ones in the *Baseline* model. In DabLog, the encoder network is connected with the decoder by *tensorflow.keras.layers.RepeatVector*, and the decoder network is connected with the classifier by *tensorflow.keras.layers.TimeDistributed*.

6.2 Experiment Setup for Traffic Logs

The UNSW-NB15 dataset [29] encloses 2.84 million log entries from a testbed that runs a IXIA PerfectStorm (traffic generator), a firewall, three servers, two routers, four attackers, and 30 endpoints, across 31 hours. All parties are identified by their IPs. Among these 30 endpoints, 10 are benign traffic initiators that tried to access the other 20 service endpoints that each ran several services. Among these 20 service endpoints, 10 are normal and 10 are victims of nine cyberattack categories, including fuzzers, penetration, backdoors, denial of service, exploits, reconnaissance, shellcode, worms, and other generic attacks. Each log entry is an aggregation of several packets that flowed from one source to one destination, and each entry contains 49 fields, including timestamps, IPs, ports, internet protocol, service type, packet counts, and byte counts. Anomaly labels (i.e., *normal* and *abnormal*) are provided at the event level, and there are 321 thousand abnormal events. Moustafa et al. [29] addressed that their labelling is done by matching processed records according to the particular NIDS scenario with transaction records.

6.2.1 ***Redefining Sequences.*** The original UNSW-NB15 dataset presents four massive sequences, and we need to split them into shorter sequences. Splitting them not only provides us with more sequences for training and testing, but also enables us to pinpoint anomalies within a shorter range. We split the four massive sequences based on source IP, destination IP, and timestamps. The split is twofold: First, different log entries that have different source-destination pairs are put into different sequences. For example, an entry for traffic from 10.0.0.1 to 10.0.0.2 and an entry for traffic from 10.0.0.1 to 10.0.0.3 are put into different sequences (hence, a sequence between two particular entities includes nothing irrelevant from the other entities); moreover, an entry for traffic from 10.0.0.1 to 10.0.0.2 and an entry for traffic from 10.0.0.2 to 10.0.0.1 are also put into different sequences (hence, sequences are directional). Second, log entries that were audited during different time periods, in terms of half-hour (30 minutes) or quarter-hour (15 minutes), are put into different sequences; for example, an entry audited at 10:01 and an entry audited at 10:31 are put into two different sequences. Let $\mathcal{H}_1$ denote the set of sequences split by half-hour interval, and $\mathcal{H}_2$ denote the set of sequences split by quarter-hour interval, then we have statistics shown in Table 2. We later demonstrate through $\mathcal{H}_1$ and $\mathcal{H}_2$ how our models are impacted by the length of sequences.

Table 2: Post-Processed UNSW-NB15 Dataset

	Training Sequences	Testing (Normal)	Testing (Abnormal)	Average Length	Max. Length
$\mathcal{H}_1$	5,000	5,039	1,166	378.95	28,971
$\mathcal{H}_2$	9,900	9,939	2,278	191.80	15,951

Within a sequence, entries are sorted chronologically based on their timestamps. We say a sequence is abnormal if it includes at least one abnormal log entry.

6.2.2 ***Dataset Engineering.*** UNSW-NB15 provides a few informative features that we can directly or easily make use of. These features include text features, binary features, and integer features. We compose our event keys by concatenating the following nine features: ***protocol*** (text), ***state*** (text), ***service*** (text), ***loopback*** (binary), ***FTP-login*** (binary), ***FTP-commands*** (integer), ***HTTP-methods*** (integer), ***transaction-depth*** (integer), and ***packet-count*** (integer). For the *packet-count* feature, we split its 1,948 distinct values into 1000-packet intervals (resulting in 13 intervals from 0-1000 to 12000-13000) and 100-packet intervals (resulting in 95 intervals). We compose event keys by concatenating the above features; for example, *"tcp,FIN,-,False,False,0,0,0,0-1000"* is the most common event key which dominates 24.16% of the dataset. Let $\mathcal{K}_1$ denote the key set that leverage 1000-packet intervals, and $\mathcal{K}_2$ denote the key set that leverage 100-packet intervals, then we have $|\mathcal{K}_1| = 366$ keys and $|\mathcal{K}_2| = 706$ keys. We later demonstrate through $\mathcal{K}_1$ and $\mathcal{K}_2$ how our models are impacted by the number of distinct event keys. Note that one benefit of applying machine learning to discrete events is that, detailed domain knowledge is no longer required. While our method of abstracting keys does not look perfect, we argue that, this method serves our purpose of evaluating the performance difference between DabLog and Baseline, as we do not intent to make them know any domain details.

6.2.3 ***Training Datasets and Testing Datasets.*** After having 5,000 and 9,000 sequences ($\mathcal{H}_1$ and $\mathcal{H}_2$) of 366 event keys and 706 event keys ($\mathcal{K}_1$ and $\mathcal{K}_2$), we then transform them into subsequences of $seqlen = 10$. We have in total more than 187 millions in the training dataset and more than 221 millions subsequences in the testing dataset The numbers of distinct subsequences (which we refer to as ***patterns***) are listed in Table 3. The training datasets include the sequences of traffics that were *"sent from"* the 10 benign traffic initiators, whereas the testing datasets include the sequences representing traffics that were *"sent to"* the 20 service endpoints. That is, in this evaluation, DabLog and Baseline learn from the send-from sequences about how to reconstruct or predict the send-to sequences. Each normal event between a traffic initiator and a service endpoint is used twice: one in the send-from sequence whose subject is the traffic initiator, and one in a send-to sequence whose subject is the service endpoint. However, in terms of normal sequences, the training datasets are different from the testing datasets, though events observed by both ends share the same nature. The difference includes loop-back traffics that were not observed by the other party, and (2) normal traffics that were sent from the four attackers (there are 39 normal subsequences from attackers).

Table 3: Statistics of Sequential Patterns of $seqlen = 10$

	$\lvert\mathcal{K}_*\rvert$ Size	Normal Patterns	Abnormal Patterns	Undecidable Patterns
$\mathcal{K}_1$	366	1,036,098	50,032	824
$\mathcal{K}_2$	706	1,660,669	52,454	532

Table 4: Metrics for Traffic Logs when $\theta = 10\%$

Model	TP	FP	TN	FN	Precision	Recall	F_1
DabLog	2145	922	9017	133	69.93%	94.16%	80.25%
Baseline	2228	3341	6598	50	40.00%	97.80%	56.77%
Frequency	2278	9939	0	0	18.64%	100.00%	31.42%

6.3 Anomaly Detection Results for Traffic Logs

To compare different models, we leverage the F_1 score metric, whose equations are listed below, where TP, FP, TN, and FN denote the numbers of true positives, false positives, true negatives, and false negatives, respectively. The higher the F_1 score, the better the model in providing good anomaly detection results. Note that, unlike prior work, we do not leverage the accuracy metric, because it is misleading for imbalanced dataset where there are way more negatives (i.e., normal subsequences) than positives (i.e., abnormal subsequences): a blind model that always returns normal for any sequences can achieve high accuracy because $TN \gg FN$.

$$\text{Precision} = \frac{TP}{TP+FP} \qquad F_1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Recall} = \frac{TP}{TP+FN} \qquad \textit{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Figure 5(a) to Figure 5(c) depict the F_1 score trends of different models upon different datasets $\mathcal{H}_1$ and $\mathcal{H}_2$, and different key sets $\mathcal{K}_1$ and $\mathcal{K}_2$. The X-axis represents the variable ranking threshold N in a normalized form $\theta_N = N/\mathcal{K}_*$, and the Y-axis represents the F_1 score of each model.

From Figure 5(a), we can see that DabLog is more advantageous for critics where fine-grained reconstruction, say $\theta_N \leq 10\%$, is used. Baseline is only slightly more advantageous when coarse-grained reconstruction, say $\theta_N \geq 25\%$, is used, but the advantage is very small. If we set the threshold at $\theta_N = 10\%$, or equivalently $N = 70$ keys, then DabLog has 2,419 less FPs, in exchange of 83 less TPs. As such, DabLogs reaches a F_1 score 80.25%, whereas Baseline can only reach 56.77%. The metrics are listed in Table 4. For reference purpose, we also include a trivial frequency model that blindly guesses top-frequent keys. Since most sequences inevitably include event keys that are not top-θ_N frequent, the frequency model simply reports every sequence abnormal when $\theta_N < 71\%$.

Figure 5(b) and Figure 5(c) further show the F_1 trends of DabLog and Baseline under different configurations (i.e., $\mathcal{K}_*$ and $\mathcal{H}_*$), and Table 5 lists their area under the F_1 curves. From them, we have three common observations for DabLog and Baseline. First, the shorter the $\mathcal{H}_*$ interval, the better the detection performance when fine-grained reconstruction is used. We can see in Figure 5(b) and Figure 5(c) that all the F_1 trends for models under the $\mathcal{H}_2$ configuration are more advantageous before $\theta_N \leq 10\%$ than the ones under $\mathcal{H}_1$. This is good, because it is easier to narrow down what is abnormal when the intervals are shorter. Second, the more event keys, the better the detection performance. Although in Figure 5(b) we see that the line for $\mathcal{K}_1$ and $\mathcal{H}_2$ reaches its plateau earlier than the line for $\mathcal{K}_2$ and $\mathcal{H}_2$, the latter has a higher plateau. In the end, having more keys (i.e., $\mathcal{K}_2$) results in DabLog getting a larger area under the F_1 curve. As shown in Table 5, having more keys with $\mathcal{K}_2$ is more advantageous than with $\mathcal{K}_1$ for both models under the same interval configuration $\mathcal{H}_*$ (the area is calculated with $0 \leq \theta_N \leq 1$ though $0.5 < \theta_N$ is not plotted). Last but not least, DabLog is more advantageous than Baseline in terms of the area under the F_1 curve under all $\mathcal{K}_*$ and $\mathcal{K}_*$ configuration combinations.

Table 5: Area Under the F_1 Curve

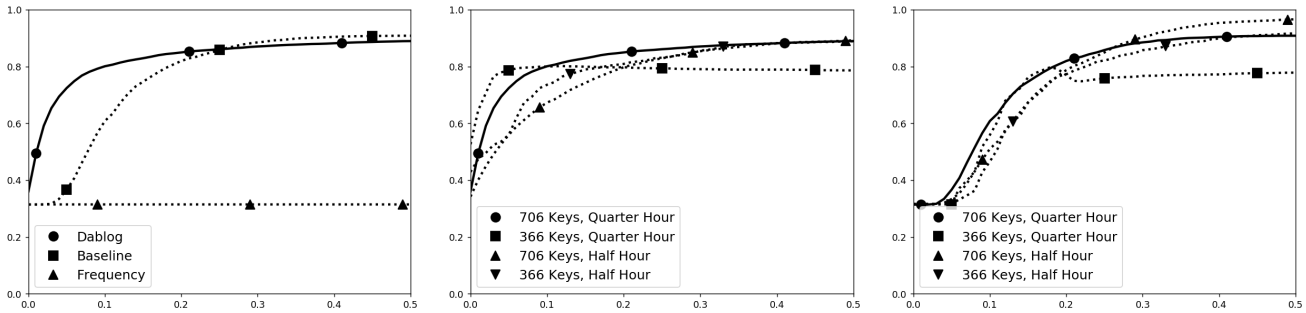
	Half-Hour Window $\mathcal{H}_1$		Quarter-Hour Window $\mathcal{H}_2$	
	$\lvert\mathcal{K}_1\rvert = 366$	$\lvert\mathcal{K}_2\rvert = 706$	$\lvert\mathcal{K}_1\rvert = 366$	$\lvert\mathcal{K}_2\rvert = 706$
DabLog	80.93%	81.72%	75.01%	83.30%
Baseline	71.76%	77.77%	63.48%	74.89%

In comparison, in terms of F_1 curves, DabLog has better performance when it has more event keys (i.e., $\mathcal{K}_2$) and shorter intervals (i.e., $\mathcal{H}_2$) compared to itself and compared to Baseline. DabLog is 8.41% more advantageous than Baseline with $\mathcal{K}_2$ and $\mathcal{H}_2$, and 3.45% more advantageous than Baseline with $\mathcal{K}_2$ and $\mathcal{H}_1$.

6.4 Case Studies

We motivate our case studies with this question: *how does the fundamental difference make DabLog more advantageous?* We discuss why DabLog is more advantageous than Baseline by studying sequential patterns. By patterns, we mean the distinct subsequences of length 10. Upon $\mathcal{K}_2$, $\mathcal{H}_2$, and $\theta_N = 10\%$, DabLog and Baseline reported 2,111 common TPs, 789 common FPs, and 16 common FNs. DabLog reported 34 exclusive TPs with trade-off 133 exclusive FPs, whereas Baseline reported 117 exclusive TPs but 2,552 exclusive FPs.

6.4.1 ***DabLog's TPs but Baseline's FNs.*** DabLog reported 34 exclusive TPs. These TPs are short sequences whose length are less than eight events (nine of which have only one events, and three of which have seven events). Sequence 1 and Sequence 2 listed in Table 6 are two examples of such sequences, sent from 175.45.176.0 (attacker) to 149.171.126.18 (endpoint). There are two instances of Sequence 1: one is a web-exploit (CVE 2014-2324), and the other is an IMAP-service buffer-overflow attack (CVE 2004-2501). Sequence 2 encloses an executable file attachment in e_4, whereas e_5 is a benign event. The events marked by asterisks (i.e., e_0 and e_3) are the events reported as anomalies by DabLog. In DabLog's point of view, *begin-of-sequence* has zero chance being before e_1 and e_4 when there are only few events happened within the 15-minute window; that is, DabLog expects events before e_1 and e_4. Top reconstruction for e_0 includes TCP entries for SSH, SMTP, and HTTP services. Contrarily, in Baseline's point of view, e_0 to e_6 are all normal, where e_1 is the most frequent event key, e_4 is the eighth frequent event key, and e_5 is the sixth frequent event key. As such, Baseline seems to be more like a frequency-based



(a) Comparison of Models upon $\mathcal{K}_2$ and $\mathcal{H}_2$ (b) DabLog with Different Configuration (c) Baseline with Different Configuration

Figure 5: F_1 Metric Comparison Among Different Models and Different Configurations (Traffic-Log Dataset)

Table 6: Example Sequential Discrete Events

#	DabLog	Baseline		Event Key
1	TP	FN	e_0*	*begin-of-sequence*
			e_1	TCP,FIN,-,False,False,0,0,0,0-100
			e_2	*end-of-sequence*
2	TP	FN	e_3*	*being-of-sequence*
			e_4	TCP,FIN,SMTP,False,False,0,0,0,0-100
			e_5	TCP,FIN,HTTP,False,False,0,1,1,0-100
			e_6	*end-of-sequence*
3	TN	FP		⋮
			e_7	UDP,CON,DNS,False,False,0,0,0,0-100
			e_8	UDP,CON,DNS,False,False,0,0,0,0-100
			e_9*	TCP,FIN,HTTP,False,False,0,4,1,1000-1100
4	FN	TP	e_{10}	*begin-of-sequence*
			e_{11}	TCP,FIN,-,False,False,0,0,0,0-100
			e_{12}	TCP,FIN,HTTP,False,False,0,1,1,0-100
			e_{13}	TCP,FIN,-,False,False,0,0,0,0-100
			e_{14}*	UDP,INT,-,False,False,0,0,0,0-100
			e_{15}	TCP,FIN,HTTP,False,False,0,1,1,0-100
			e_{16}	TCP,FIN,-,False,False,0,0,0,0-100
			e_{17}	TCP,FIN,HTTP,False,False,0,1,1,0-100
			e_{18}	TCP,FIN,-,False,False,0,0,0,0-100
			e_{19}*	UDP,INT,-,False,False,0,0,0,0-100

model. Its fundamental limitation is that Baseline cannot foresee upcoming events, and hence it has less information to correctly judge a sequence when only few information is seen.

6.4.2 ***DabLog's FPs.*** DabLog reported 922 FPs, 789 of which are exclusive FPs (that Baseline did not report) and 133 of which are common FPs (that Baseline also reported). We studied these FPs, and we found out all these FPs were of rare patterns. Recall that, we have in total 1.66 million normal patterns (Table 3), and it is infeasible to memorize all of them considering the limited numbers of LSTM cells. Each FP pattern (out of 3746 patterns in the 789 exclusive FPs) has only zero or one occurrence in the training set, and each pattern (out of 432 patterns in the 133 common FPs) has a maximum of two occurrences in the training set. Their rarity makes them very hard to learn. The factor that caused 789 different judgements (DabLog's FPs but Baseline's TN) is, again, event key frequency. The top three abnormal keys that both DabLog and Baseline reported abnormal (in common FPs) each has 34, 44, and 50 occurrences (in the entire dataset of size 2.84 million events), respectively. Baseline reported them abnormal because they are extremely rare events. In contrast, the top three abnormal keys that DabLog exclusively reported abnormal (in the 789 exclusive FPs) are the top-8th, 13th, and 32th frequent keys, respectively. Baseline did not report them abnormal because these frequent keys are in Baseline's top $\theta_N = 10\%$ predictions due to their frequency. While Baseline has these 789 exclusive TNs, it has a tradeoff of 2,552 exclusive FPs due to false prediction based on frequency.

6.4.3 ***DabLog's TNs but Baseline's FPs.*** Baseline reported 2,552 exclusive FPs, and all these sequences are long sequences, with average length of 203 events. The longer a sequence, the more likely it includes a rare key that will cause Baseline to produce FPs. Sequence 3 listed in Table 6 is an example, in which Baseline reported e_9 as an anomaly. In the sequence, e_9 is a HTTP entry that follows two DNS entries, and e_9 is an event key has 430 occurrences in the entire dataset. Events before e_7 include other DNS entries and SSH entries. Baseline reported e_9 as an anomaly because e_9 is at rank 15.46%, while in DabLog's point of view e_9 is at rank 3.4%. DabLog not only can reconstruct this subsequence but also all 2,552 exclusive FPs of Baseline, or equivalently 493,285 subsequences of 2,550 patterns. These 2,550 patterns together have 6,441 occurrences in our training dataset, and DabLog has learned how to reconstruct them. The fundamental difference between DabLog and Baseline that causes these 2,552 different judgements is that, DabLog sees sequences as atomic instances and tries to reconstruct them, whereas Baseline sees events as atomic instances and tries to *"predict"* them based on what subsequences Baseline has seen. With this fundamental difference, DabLog is more advantageous not only in finding TPs within short sequences, but also in avoiding FPs within long sequences.

6.4.4 ***DabLog's FNs.*** DabLog reported 133 FNs, 117 of which are exclusives FNs (that Baseline did not report) and 13 of which are common FNs (that Baseline also reported). Among the 133 exclusive FNs, there are 34,802 occurrences of 113 abnormal patterns. We studied them and found out that these 133 abnormal patterns are also in our training set as normal patterns. The way we engineered event keys cannot avoid two traffics (one benign and one malicious) having a same subsequence of length 10. There are 1,965

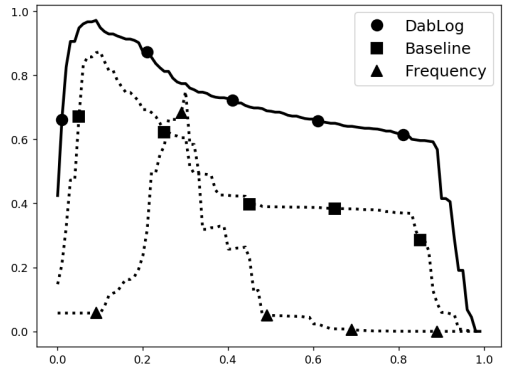


Figure 6: F_1 Metric Comparison (System-Log Dataset)

occurrences of these 113 abnormal patterns in our training dataset. They are so frequent that DabLog learned how to reconstruct them (and Baseline learned to predict 12 out of 113 abnormal patterns). It is similar to the dataset poisoning problem, which is a common challenges to machine-learning techniques for anomaly detection. Forcibly reporting any of these 113 abnormal patters as anomalies will inevitably cause massive amount of FPs, as they have 3,930 normal occurrences in our testing set. The factor that causes 117 different judgements (DabLog's FNs but Baseline's TP) is, again, event-key frequency, and a case study is outlined below.

DabLog's FNs but Baseline's TP: In Table 6, Sequence 4 is an example sent from 175.45.176.3 to 149.171.126.18. It is a combination of reconnaissance, remote-code execution, buffer-overflow, DoS, and race-condition exploits. The events marked by asterisks (i.e., e_{14} and e_{19}) are the events reported as anomalies by Baseline, and they are of the same event key which is the 12th most-frequent key. Baseline reported them as anomalies because they are not within $\theta_N = 10\%$ threshold, with e_{14} ranked at 13.74% and e_{19} ranked at 13.74%. The events e_{14} and e_{19} are in-fact *SunRPC UDP Portmapper GETPORT Requests*. These events are labeled as *reconnaissance* but did not do anything truly malicious. Although Baseline correctly reported Sequence 4, it is not based on attacks or exploits but key frequency. Top predictions include top-10 most-frequent keys, and the 11th key has less than 1% probability in its predictions. The fundamental limitation is that, again, when Baseline only has few information, it can only blindly guess upcoming events. There are many similar instances where events are reported as anomalies because it did not see sufficient information but event keys are not top frequent. As such, Baseline reported 2,552 exclusive FPs.

6.5 Evaluation Summary for System Log

The HDFS dataset [39] encloses 558,223 normal sessions and 16,838 abnormal sessions. To measure how well the related work can be applied to scenarios where more event keys are involved, we re-crafted the event keys into $\mathcal{K}'$. With $|\mathcal{K}'| = 101$ and $seqlen = 10$, the post-processed HDFS dataset has in total 276,499 patterns, 220,912 of which are normal patterns and 35,662 are abnormal patters. The training dataset consists of 200,000 normal sessions, and the testing datasets consists of non-overlapping 200,000 normal sessions and 16,868 abnormal sessions. Figure 6 depicts the F_1 score trends. DabLog has its peak F_1 score 97.18% at $\theta_N = 9\%$, whereas Baseline has its peak F_1 Score 87.32% at $\theta_N = 10\%$. We can see that DabLog is more advantageous for critics where fine-grained reconstruction, say $N \leq 10$, is used. Previous work [10, 40] was also evaluated by the area under Receiver Operating Characteristic (ROC). DabLog achieves an area of 99.44%, whereas Baseline achieves an area of 97.23%. We can see that DabLog is more advantageous.

7 DISCUSSION AND FUTURE WORK

7.1 A More Comprehensive Embedding

Our embedding layer has the drawback of handling *unknown event keys* that are not in the training data but in the testing data. In the literature of natural language processing, this problem is also known as the *out-of-vocabulary* problem, and there are two common workaround options for it. One option is to substitute the designated *"unknown"* word for not only unknown words but also rare words, so that *"unknown"* is trained as if it is some known rare events. This option, however, is not applicable to security audit logs, because unknown events can be frequent and similar to known events (e.g., *read X* is similar to *read A*), and wrongly treating unknown events as rare events may cause FPs. The other option is to leverage a pre-trained Word2Vec embedding in building a new embedding model (e.g. Mimick embeding [33]). Inspired by the latter option, one of our future work is to build an Event2Vec embedding model, which can derive the embedding by examining words in k_*.

7.2 A Double-Threshold Criterion

We presented DabLog's critic as a rank-based criterion in order to compare DabLog with existing work [10, 40]. However, both rank-based and threshold-based criteria have the same major drawback: they brutally divide the probabilistic distribution P_τ into two parts (normal and abnormal), while omitting the correlation between the true $\hat{k}_i$ and the top-N prediction keys k_j. Let us consider a case in which $|p_i - p_j|$ is small and $|\mathcal{E}(\hat{k}_i) - \mathcal{E}(k_j)|$ is small, meaning that the corresponding keys $\hat{k}_i$ and k_j are almost equivalent to the model, still $\hat{k}_i$ could be abnormal while k_j is normal if the pivotal point sits in between (hence a FP). If a criterion can also examine the underlying correlation between $\hat{k}_i$ and k_j, that is $|\mathcal{E}(\hat{k}_i) - \mathcal{E}(k_j)|$, such a FP can be avoided. One of our future work is to study the performance of such a *rank-distance double-threshold criterion*).

7.3 DabLog is beyond a Standard Autoencoder

We would like emphasize again that, although DabLog is based on the autoencoder methodology, DabLog is more than a standard autoencoder. Compared to the reconstruction problem for standard autoencoders, our problem—*identifying time-sensitive anomaly by examining discrete events*—is more like a language processing problem. We can view sequences S as sentences and events e_t as words, and we care more about wording (e.g., *"which words e_t better fit in the current sentence S"*) rather than embedding (e.g., *"which vector y_τ better vectorize e_t in the current sentence S"*). As such, DabLog is designed as an *embed-encode-decode-classify-critic* model, so that it can help us with finding *"fitting words"* and *"unfitting words"*, or equivalently, normal events and abnormal events in the sequential context of S. In contrast, typical time-insensitive autoencoder-based

anomaly detection methods directly use scalar reconstruction errors (e.g., root-mean-square error) as anomaly scores.

8 CONCLUSION

With regard to anomaly detection approaches for discrete events, we address a fundamental limitation of the widely adopted predictor-based methodology through our in-depth case studies. We argue that recomposing sequences is also an attractive methodology, especially in real-world contexts where the need of detection with more keys cannot be well satisfied by predictor-based methodology. We propose DabLog and evaluate DabLog with UNSW-NB15 dataset and HDFS dataset. Our results showed that DabLog outperforms our predictor-based baseline model in terms of F_1 score. With reconstruction of sequential events, DabLog not only introduces much fewer FPs, but also improves awareness regarding *what is normal* in contexts that involve more keys.

ACKNOWLEDGMENTS

Peng Liu was supported by ARO W911NF-13-1-0421 (MURI), NSF CNS-1814679, and NSF CNS-2019340.

REFERENCES

[1] Mejbah Alam, Justin Gottschlich, Nesime Tatbul, Javier Turek, Timothy Mattson, and Abdullah Muzahid. 2017. A Zero-Positive Learning Approach for Diagnosing Software Performance Regressions. arXiv:1709.07536 [cs.SE]

[2] Arwa Aldweesh, Abdelouahid Derhab, and Ahmed Z. Emam. 2020. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems* 189 (2020), 105124.

[3] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.

[4] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. 2018. Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems* (Tempe, AZ, USA) *(MLCS'18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 8 pages.

[5] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. 2017. Robust, Deep and Inductive Anomaly Detection. In *Machine Learning and Knowledge Discovery in Databases*, Michelangelo Ceci, Jaakko Hollmén, Ljupčo Todorovski, Celine Vens, and Sašo Džeroski (Eds.). Springer International Publishing, Cham, 36–51.

[6] Zouhair Chiba, Noureddine Abghour, Khalid Moussaid, Amina El Omri, and Mohamed Rida. 2018. A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection. *Computers & Security* 75 (2018), 36 – 58.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014). arXiv:1406.1078

[8] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong Anomaly Detection Through Unlearning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1283–1297.

[9] M. Du and F. Li. 2016. Spell: Streaming Parsing of System Event Logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 859–864.

[10] 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* (Dallas, Texas, USA) *(CCS '17)*. ACM, New York, NY, USA, 1285–1298.

[11] M. O. Ezeme, Q. H. Mahmoud, and A. Azim. 2018. Hierarchical Attention-Based Anomaly Detection Model for Embedded Operating Systems. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 225–231.

[12] O. M. Ezeme, Q. H. Mahmoud, and A. Azim. 2019. DReAM: Deep Recursive Attentive Model for Anomaly Detection in Kernel Events. *IEEE Access* 7 (2019), 18860–18870.

[13] Filipe Falcão, Tommaso Zoppi, Caio Barbosa Viera Silva, Anderson Santos, Baldoino Fonseca, Andrea Ceccarelli, and Andrea Bondavalli. 2019. Quantitative Comparison of Unsupervised Anomaly Detection Algorithms for Intrusion Detection. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (Limassol, Cyprus) *(SAC '19)*. Association for Computing Machinery, New York, NY, USA, 318–327.

[14] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. arXiv:1303.5778 [cs.NE]

[15] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A Search Space Odyssey. *CoRR* abs/1503.04069 (2015). arXiv:1503.04069

[16] Michiel Hermans and Benjamin Schrauwen. 2013. Training and Analysing Deep Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 190–198.

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[18] Q. Hu, B. Tang, and D. Lin. 2017. Anomalous User Activity Detection in Enterprise Multi-source Logs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 797–803.

[19] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International conference on machine learning*. 2342–2350.

[20] Tayeb Kenaza, Khadidja Bennaceur, and Abdenour Labed. 2018. An Efficient Hybrid SVDD/Clustering Approach for Anomaly-Based Intrusion Detection. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 435–443.

[21] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. 2019. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1777–1794.

[22] L. Liu, C. Chen, J. Zhang, O. De Vel, and Y. Xiang. 2019. Insider Threat Identification Using the Simultaneous Neural Learning of Multi-Source Logs. *IEEE Access* 7 (2019), 183162–183176.

[23] Liu Liu, Chao Chen, Jun Zhang, Olivier De Vel, and Yang Xiang. 2019. Unsupervised Insider Detection Through Neural Feature Learning and Model Optimisation. In *Network and System Security*, Joseph K. Liu and Xinyi Huang (Eds.). Springer International Publishing, Cham, 18–36.

[24] L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang. 2018. Anomaly-Based Insider Threat Detection Using Deep Autoencoders. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. 39–48.

[25] Z. Liu, T. Qin, X. Guan, H. Jiang, and C. Wang. 2018. An Integrated Method for Anomaly Detection From Massive System Logs. *IEEE Access* 6 (2018), 30602–30611.

[26] X. Lu, W. Zhang, and J. Huang. 2020. Exploiting Embedding Manifold of Autoencoders for Hyperspectral Anomaly Detection. *IEEE Transactions on Geoscience and Remote Sensing* 58, 3 (March 2020), 1527–1537.

[27] Marcus A. Maloof and Gregory D. Stephens. 2007. ELICIT: A System for Detecting Insiders Who Violate Need-to-Know. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection* (Gold Goast, Australia) *(RAID'07)*. Springer-Verlag, Berlin, Heidelberg, 146–166.

[28] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. arXiv:1802.09089 [cs.CR]

[29] N. Moustafa and J. Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*. 1–6.

[30] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan. 2019. GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*. 91–99.

[31] A. Oprea, Z. Li, T. Yen, S. H. Chin, and S. Alrwais. 2015. Detection of Early-Stage Enterprise Infection by Mining Large-Scale Log Data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 45–56.

[32] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to Construct Deep Recurrent Neural Networks. arXiv:1312.6026 [cs.NE]

[33] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking Word Embeddings using Subword RNNs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, 102–112.

[34] Mayu Sakurada and Takehisa Yairi. 2014. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (Gold Coast, Australia QLD, Australia) *(MLSDA'14)*. Association for Computing Machinery, New York, NY, USA, 4–11.

[35] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. 2018. A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, 1 (Feb 2018), 41–50.

[36] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised Learning of Video Representations using LSTMs. *CoRR* abs/1502.04681 (2015). arXiv:1502.04681

[37] Xuhong Wang, Ying Du, Shijie Lin, Ping Cui, Yuntian Shen, and Yupu Yang. 2020. adVAE: A self-adversarial variational autoencoder with Gaussian anomaly prior knowledge for anomaly detection. *Knowledge-Based Systems* 190 (2020), 105187.

[38] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online System Problem Detection by Mining Patterns of Console Logs. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining (ICDM '09)*. IEEE Computer Society, USA, 588–597.

[39] 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* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 117–132.

[40] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang. 2019. nLSALog: An Anomaly Detection Framework for Log Sequence in Security Management. *IEEE Access* 7 (2019), 181152–181164.

[41] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula. 2017. Autoencoder-based feature learning for cyber security applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 3854–3861.

[42] Chong Zhou and Randy C. Paffenroth. 2017. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 665–674.

[43] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In *International Conference on Learning Representations*.