Hunting for Insider Threats Using LSTM-based Anomaly Detection

Miguel Villarreal-Vasquez,Gaspar Modelo-Howard, *Senior Member, IEEE,* Bharat Bhargava, *Fellow, IEEE,* and Simant Dube

Abstract—Insider threats are one of the most difficult problems to solve, given the privileges and information available to insiders to launch different types of attacks. Current security systems can record and analyze sequences from a deluge of log data, potentially becoming a tool to detect insider threats. The issue is that insiders mix the sequence of attack steps with valid actions, reducing the capacity of security systems to programmatically detect the attacks. To address this shortcoming, we introduce LADOHD, an anomaly detection framework based on Long-Short Term Memory (LSTM) models, which learns the expected event patterns in a computer system to identify attack sequences even when attacks span for a long time. The applicability of the framework is demonstrated on a dataset of 38.9 million events collected from a commercial network of 30 computers over twenty days and where a 4-day long insider threat attack occurs. Results show that LADOHD is able to detect anomalies generated by the attack with a True Positive Rate of 97.29% and False Positive Rate of 0.38%. LADOHD outperforms the endpoint detection system used to protect the commercial network, as well as frameworks based on other methods like Hidden Markov Models.

Index Terms—Anomaly detection, endpoint detection and response (EDR), high-dimensional data, insider threats, long short-term memory (LSTM), order-aware recognition (OAR) problem, sequence analysis, variable-length system activity event sequences.

1 Introduction

A Demanding challenge for security systems is to successfully defend against insider threats because insiders are in possession of credentials, have (some) knowledge of the system operation, and are implicitly trusted as members of the organization [1]. They are also located inside the security perimeter, allowing them to unsuspiciously deploy attacks such as data exfiltration, tampering with data, and deletion of critical data [2], [3]. They commonly use sophisticated strategies to avoid detection like those in multistage persistent threats [4] and mimicry attack [5], [6], [7]. Namely, insiders mix malicious event sequences with benign actions to exploit the incapacity of defensive systems to discern event sequences after certain length, which is referred to as the *order-aware recognition* (OAR) problem [8]. Existing enterprise protection systems endeavor to counter this increased sophistication in insider evasion attacks through the application of anomaly detection methods based on advanced machine learning. Machines in customer companies run Endpoint Detection and Response (EDR) agents that generate high volumes of system events that are examined through centralized analytics modules running at the security-provider company. The ultimate goal is to detect stealthy threats, including zero-day exploits, by analyzing patterns and relationships of the aggregated data collected from these multiple endpoints at runtime. In current enterprise solutions, many of the collected malicious events are correctly classified as alerts. However, others are ignored and considered benign events despite being part of the attacks that span for a long period of time. These undetected malicious events are usually related to those detected and identified as alerts, but they are missed because of the lack of optimal solutions able to find the existing relationships among distant events in a sequence. This brings the need for precise system behavior modeling capable of capturing long-range relationships (i.e., long term dependencies) in multiple context for event sequence analysis and detection of anomalies at runtime [9].

The paradigm of anomaly detection [8], [10], [11], [12], [13], [14], [15], [16] involves the construction of patterns of normal behavior of systems and deems as anomalous (or possible intrusion) any action that does not conform to the learned patterns [17], [18]. Prior research work have been devoted to investigate and develop anomaly detection systems using sequence analysis strategies. Some of these detection techniques are based on n-gram [17], [19], [20] and others on Hidden Markov Model (HMM) [9], [17], [21], [22], [23], [24], [25]. In general, these techniques learn observed patterns in a training phase and identify as anomalous event sequences that deviate from them during testing. In particular, HMM-based methods estimate the likelihood of events conditioned on some number of previous events (e.g., after observing $n-1$ previous events). This allows determining not only whether a sequence of certain length (i.e., n in this case) is feasible to occur, but also how likely it occurs in normal (non-attack) conditions. However, Yao et al. [8] presented a comprehensive analysis of these techniques and showed they are incapable to discern the order of events in long sequences due to the OAR problem, restricting the

- *This work was supported by grants from the Northrop Grumman Cybersecurity Research Consortium (NGCRC).*
- *M. Villarreal-Vasquez and B. Bhargava are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA (email: mvillar@purdue.edu; bbshail@purdue.edu).*
- *G. Modelo-Howard is with Palo Alto Networks, Santa Clara, CA 95054 USA (email: gaspar@acm.org).*
- *S. Dube is with Broadcom Inc., Mountain View, CA 94043 USA (email: simant.dube@broadcom.com).*

Manuscript received June 18, 2020.

length n of the analyzed sequences to small values.

In this paper, we present a LSTM-based anomaly detection framework that collects and analyzes high volumes of system events from multiple distributed EDR agents to protect against insider threats at runtime. We refer to the framework as LADOHD (LSTM-based Anomaly Detector Over High-dimensional Data) due to the high feature dimensionality of the produced events. LADOHD tackles the OAR problem by leveraging the event relationship information extracted from different endpoints as well as the properties ingrained to LSTMs and its variants [26], [27], such as memory, short and long term dependencies, stateful representation, and capacity to process variable length sequences [28], [29]. We hypothesize that these properties give these models the ability to detect variable-length anomalous sequences and the potential to recognize attacks deployed by insiders that span for a long time. Specifically, our LSTM-based technique answers the anomaly detection problem of given a sequence of events $e_1, e_2, \ldots, e_{n-1}$, whether or not the sequence $e_1, e_2, \ldots, e_{n-1}, e_n$ should occur. Our technique operates with variable values of n and detects non-conforming patterns with respect to the learned models by analyzing the event sequences formed by system activities. Each possible system activity is enumerated and uniquely identified to form the vocabulary of system events. At any time t, our detector computes the probability of each possible event to be the next one given the previous sequence of events observed until time t-1. The detection is then made by analyzing the distribution of these probability values.

The obtained results include quantitative measurements of the detection capacity of the proposed technique tested over a dataset of 38.9 million activity events. These events were collected from multiple security endpoints running on more than 30 machines for 28 days. It is shown through different experiments that our framework successfully achieve detection with a TPR and a FPR of 97.29% and 0.38% respectively. Below, our research contributions:

- We implement a prototype of LADOHD [30] evaluated with a dataset of 38.9 million activity events collected from an enterprise EDR system. Results show that our method achieves a high detection rate above 97% while keeping a FPR $< 0.5\%$. Furthermore, it is shown that LADOHD detected more malicious events under the same attack than the EDR of the same company currently in production.
- A deep analysis of the features of the events generated by the EDR system is presented. Feature were selected to form a vocabulary of events that allow the model successfully learning long-term dependencies.
- We measure how far LSTM-based models look backward to rank probable events in each timestep of a sequence. We demonstrate that LSTMs have a better capacity than alternative methods (e.g., HMM-based methods) to solve the OAR problem.
- We are the first presenting a comprehensive analysis of the strengths, limitations and applicability of LSTM-based models to counter insider threats via anomaly detection in real-word scenarios.

2 Overview and Threat Model

2.1 Overview

LADOHD builds LSTM-based behavioral profiles of applications using the system event sequences collected from multiple endpoints running a renowned EDR agent. Its goal is to detect anomalous or non-conforming execution patterns at runtime in two phases. First, a training or observation phase, in which the profile of a selected application is built by learning the relationships among events in patterns or sequences observed when the application runs in normal (non-attack) conditions. Second, a testing or evaluation phase, in which the learned model is used to estimate the probability of each possible event to be the next event in a sequence given the sequence of previous events. In the latter phase, low probable events are classified as anomalous.

We assume that the generated event sequences follow a well-structured pattern (e.g. execution path of programs) with a consistent relationship among events. Consequently, the resulting sequences are thought as an structured language that can be analyzed using LSTM-based models as it has been done via Natural Language Processing (NLP) to solve problems such as language modeling (i.e., prediction of the next word in a text) [31], [32].

LADOHD requires the definition of a finite set of possible symbols $E = \{1, 2, ..., N\}$, which corresponds to all the possible events related to the application of interest that are considered in the detection process (hereafter, we will refer to this set as vocabulary of events). At training, LADOHD extracts all the subsequences containing the events in E from the set of event sequences $S = \{s_1, s_2, ...s_N\}$ generated by N endpoints. These subsequences are used to train the LSTM-Based model.

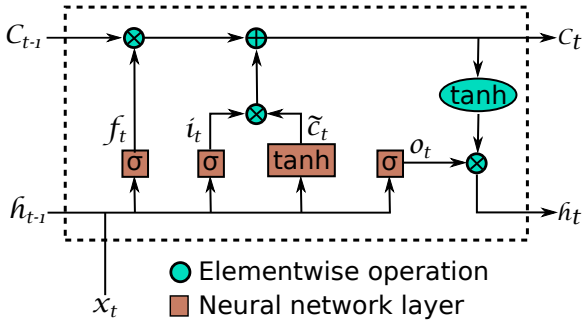
The definition of the vocabulary of events E is crucial because there is a trade-off between the granularity of the events and the number of unseen events that appear it the evaluation or testing phase. For our experiments, we defined E in such a way that most of the events observed at training are also observed at testing, reducing the number of unseen events during the evaluation phase. Section 4 includes the details of our definition.

During the evaluation phase, given a previous sequence of events until timestep $t-1$ $e_1, e_2, ..., e_{t-1}$ ($e_i \in E$), the trained model outputs an array of probabilities of length $|E|$, representing the probabilistic estimation of each event in E to be the next event at timestep t. For the detection, LADOHD uses this output and finds the set K of the top k most likely events to occur at time t. When an event $e_t \in E$ is observed at time t, it is considered benign if $e_t \in K$, anomalous otherwise.

For any sequence $s = e_1 e_2 ... e_{t-1}$, our framework computes the probabilities of possible events next in the sequence $P(e_i|e_{1:i-1})$ for $i = 1, 2, ...$ This versatile approach allows not only validating each event at runtime, but also estimating the probability the entire sequence s by applying the chain rule as shown in Equation 1.

$$P(s) = \prod_{i=2}^{t} P(e_i|e_{1:i-1}) \tag{1}$$

LADOHD operates with system events collected from multiple monitored machines. The EDR agent running in these



(a) LSTM Cell

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$
$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$
$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$
$$\tilde{c}_t = tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot tanh(C_t)$$

(b) Operations in the LSTM Cell

Fig. 1. Architecture of a LSTM cell. Figure 1a shows the internal connections of the different components and the output of the internal operations of the cell. Figure 1b shows the details of these operations performed over the current input and previous hidden state to compute the current output.

machines generates an event for every activity conducted by a specified process (whether malicious or not). Each event includes a comprehensive set of information about the *actor* (process executing the action), detailed description of the *action*, and information about the *target* (object over which the action is executed). The pieces of information considered during the monitoring process and their interpretation define the vocabulary of events and its granularity. For example, consider the scenario where a "process A (actor) connects (action) to specific IPv4 address X.X.X.X (target)." This event might be defined as "A connects X.X.X.X", where X.X.X.X represents any possible IPv4 address, producing a vocabulary with high granularity. The same event, however, might be defined as "A connects X", with X being either 0 or 1 to represent whether the IPv4 address is internal or external respectively. In the latter case, due to the low granularity, the vocabulary size is significantly reduced.

2.2 Threat Model

We consider an insider threat who launches a multistage advance persistent attack. The insider is assumed knowledgeable in computer security and is initially assigned non-administrative privileges in a local machine. The goal of the attacker is stealing information by executing multiple steps, including a user escalation followed by a data exfiltration phase. The insider initially exploits already installed applications such as Powershell and runs malicious scripts to establish remote connections to send the stolen data.

3 Background and Related Work

3.1 LSTM Networks

LSTMs are a type of recurrent neural network (RNN) able to learn long-term dependencies (i.e., relationships between elements in distant positions of a sequence) [37]. They achieve this goal through a complex memory structure in the LSTM cell not included in traditional RNN cells. Figure 1 shows this structure. The matrices and vectors W_x, U_x, and b_x (with $x \in \{f, i, c, o\}$) in Figure 1b are the parameters θ of a LSTM [37]. Their interaction is shown in Figure 1a. Like traditional RNNs, LSTMs process each input at time t along with the output of the previous timestep (h_{t-1}). In addition, they include a unit called cell state (C) that carries on information of the entire sequence. LSTMs adds to or removes minor pieces of information from C through the operations in the forget (f_t) and input (i_t) gates. The new C represents the event history used to compute the output h_t by filtering C out with the output gate o_t. This architecture gives LSTMs the capacity to relate current events with distant past events in a sequence, making them suitable to detect anomalies produced by insider threat activities.

LSTMs can be implemented as multi-class classifiers that map a m-dimensional input symbol $x \in \mathbb{R}^m$ into one of n classes (each class corresponding to one of the the possible events). The output of a LSTM (with a $softmax$ [38] layer at the end) is a n-dimensional tensor $y \in \mathbb{R}^n$, which represents the probability distribution of the n classes. Namely, the element y_i of the output y represents the probability that input x corresponds to the class i. To operate as a sequential multi-class classifier, LSTMs are trained using backpropagation [39] with a set of pairs (x, y), where x is an input sequence of classes and y the expected next class of the sequence x. The training pair (x, y) is customized to control the timestep windows (w) used to update the parameters θ. For example, given a input sequence $x = x_1, x_2, ..., x_{t-1}$, the network can be trained to predict either the element x_t ($w = t - 1$) or each of the elements $x_2, ..., x_{t-1}, x_t$ ($w = 1$). When $w = 1$ (our case as described in Section 4.4), the LSTM outputs the probability $P(x_t|x_{1:t-1})$ at each timestep t, which allows classifying low probable events as anomalous regardless the length of the previous sequence.

3.2 Order-Aware Recognition (OAR) Problem

The OAR problem is an anomaly detection problem that refers to the incapacity of distinguishing sequences after certain length [8]. Given a ordered sequence of events *abcba* the corresponding set of 2-tuple adjacent events is {*ab*, *bc*, *cb*, *ba*}. The same set results from these other two ordered sequences *cbabc* and *bcbab*. As the 2-tuple adjacent event set is the same for these three ordered sequences of the example, methods able to analyze sequences of length 2 or less cannot discern among these ordered sequences. This can be better observed if the 3-tuple adjacent events of the sequences *abcba*, *cbabc* and *bcbab* are considered, which respectively are {*abc*, *bcb*, *cba*}, {*cba*, *bab*, *abc*} and {*bcb*, *cba*, *bab*}. Clearly, in this case methods able to analyze sequences of length 3 can distinguish the three ordered sequences *abcba*, *cbabc* and *bcbab* as the resulting sets are different. We investigate

TABLE 1
Comparison With Existing LSTM-based Security Solutions

Research	Anomaly Detection	Benign Data Only	LSTM Only	Basic Analysis	Extended Analysis
System Call Language Modeling [33]	✗	✗	✗	✗	✗
Multi-level Detector (For ICS) [34]	✓	✓	✗	✗	✗
Deeplog [35]	✓	✓	✓	✗	✗
Tiresias [36]	✗	✗	✓	✓	✗
LADOHD [this work]	✓	✓	✓	✗	✓

how feasible and until what extend LSTM-based models can solve the OAR problem. This is an unsolved question and one of our main contributions.

3.3 Endpoint Detection and Response

Endpoint Detection and Response (EDR) systems work by monitoring endpoint and network activity and storing the corresponding logs in a central database where further analysis and alerting takes place. An EDR agent is installed in each of the protected endpoints, acting as the first line of defense against attacks and providing the foundation for event monitoring and reporting across the network. EDR systems evolved from malware protection solutions, as software vendors added data collection and exploration capabilities, thanks to the increasing computing and storage capacity of the hosts where the agents run. The present challenge for EDR systems is to significantly increase its detection capabilities from the vast amounts of data collected, especially for attacks that are recorded as long sequences like those deployed by insider threats.

3.4 Anomaly Detection Based on Sequence Analysis Using Non-LSTM Approaches

The methods presented in this section proposed sequence analysis as an anomaly detection mechanism to detect control-flow violations.The methods build behavioral models based on n-gram and n-order HMM to detect unseen or low probable patterns.

Anomaly detection methods based on n-gram [19], [20] work by enumerating all observed sequences of length n (n-grams) to subsequently monitor for unknown patterns. The scalability problem of these methods (impossibility of listing all possible sequences and high false positive rate) is described by Warrender et al. [17], who proposed an alternative frequency based method. In this new method each n-gram is assigned a probability to form a histogram vector corresponding to a point in a multidimensional space. At evaluation time, the similarity of a new sequence of length n (represented as a vector) with respect to the observed points is estimated to determine whether the sequence is anomalous. Despite its improvement in scalability, this approach and the previous enumerating based method were proved to be effective for small value of n only (e.g., 3–15), making them not convenient for the detection of attacks consisting of long sequences [8].

Other previous work [9], [21], [22] focused on the application of n-order HMM to probabilistically determine how feasible a sequence of system events is. In [21], a comparison of different hidden states configuration of first-order HMM ($n = 1$) for anomaly detection is presented. It was found that both configurations full connected HMM (i.e., number of hidden states equal to the number of all possible events), and a left-to-right HMM (i.e., number of hidden states corresponds to the length of the training sequences) provide similar results differing mainly in the required training time. Results, although, show the efficiency of both configurations is significantly low having in some cases a TPR of only 55.6%. The other two HMM based methods [9], [22] use a first-order full connected HMM to detect anomalous sequences of system or library calls. These methods are similar to the one described in [21], with the addition of a new HMM initialization approach for the transition, emission and initial probabilities. The information for the initialization is extracted through static analysis of the programs. With this strategy, the results show a significant improvement in the TPR. All the described HMM based methods [9], [21], [22] applied the dynamic programming algorithm Viterbi [40] for inference. The time complexity of this algorithm is $O(|S|^2)$, with S being the set of hidden states [41]. As the lengths of the sequences to be processed by these methods depend on the number of states used in the configuration, this scalability issue restricts these methods to operate over short event sequences only.

3.5 Anomaly Detection Based on Sequence Analysis Using LSTM

Some research work havs endeavored to investigate the application of LSTM-based models to anomaly detection and similar security problems [33], [34], [35], [36]. In essence, these approaches work based on the same assumptions described in Section 2.1. Although this prior work proved the efficiency of LSTMs to accurately estimate the likelihood of a given event sequence, their ability to solve the order-aware recognition problem and their potential against modern evasion attacks seems not to have received much attention.

Kim et al. [33] present an ensemble method of LSTM models followed by threshold-based classifiers for intrusion detection in flows of system calls. The resulting ensemble is trained in a supervised manner (with both benign and malicious sequences) to classify sequences as either normal or anomalous. Obtained results are compared with other classifiers such as k-nearest neighbor (kNN) and k-means clustering (kMC). Details of neither the impact of sequence lengths nor properties of LSTM models on the detection process are included.

In [34] a multi-level approach for anomaly detection in Industrial Control Systems (ICS) is proposed. It consists of a bloom filter to discard events not seen during the training phase followed by a LSTM layer to detect unexpected

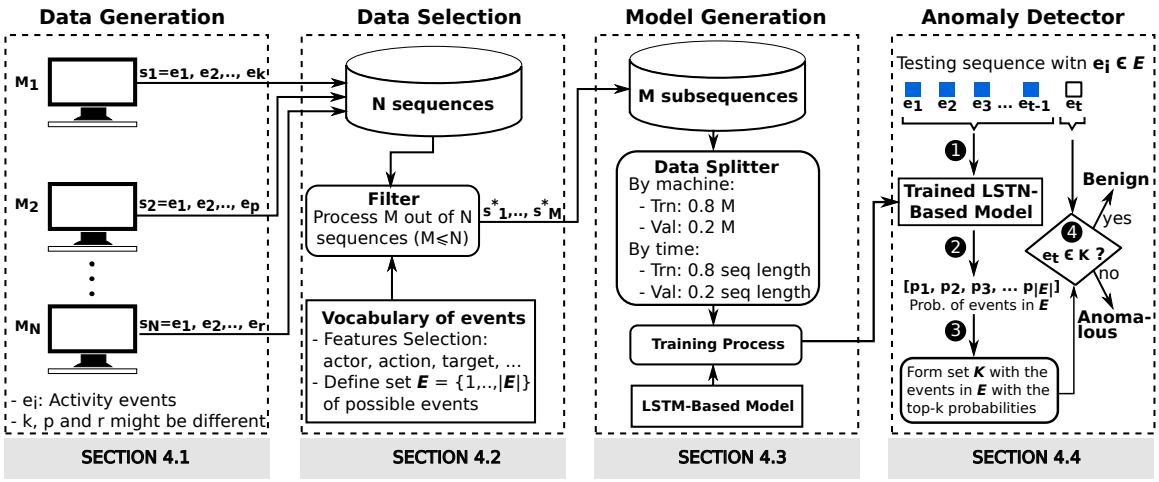


Fig. 2. Components of our anomaly detection framework LADOHD to counter insider threats: (1) data generation, (2) data selection, (3) model generation, and (4) anomaly detector. Below each component, there is a reference to the section providing a detailed explanation about its operation.

events. An event is considered unexpected or anomalous if its probability is not among the top-*k* output probabilities of the model (*k* being an adjustable parameter). The LSTM layer of the detector is trained with non-malicious data only. Results of the two layers combined are reported without further analysis about the LSTM model itself and its impact on the efficiency of the detector.

Du et al. [35] developed Deeplog, a technique to find anomalies using information available in system logs. To that end, a LSTM model is trained using a small portion of the non-malicious data to determine the next action to occur given a previous sequence of actions. Actions are conducted using different parameter values in each occurrence (e.g. files, execution time, etc.). For each identified action, a different LSTM model is trained using the sequence of observed parameter values. The goal is using these multiple models for not only determining the set of expected actions to occur, but also validating the probability of the parameter value used in that action. The model for prediction of actions is trained using a sliding length window h. Namely, given a sequence of h actions, the model predicts the $h+1$ action. The window moves forward one step at at time during training. At testing, the probability of each sliding $h+1$ action in the tested sequence is estimated. The sequence is considered anomalous if there exists at least one action in it whose probability is not among the top-*k* most likely output probabilities of the model. The experiments were conducted with small values of h (e.g. 10), and were not focused on determining the limits of LSTM models with respect to the length of the sequences.

A more recent work, called TIRESIAS [36], uses LSTM-based models for attack intend prediction. That is, the technique predicts the next step in an attack given the previous sequence steps. The dataset used in this research comprises events generated by security agents installed in thousands of machines. The sequences generated by 80% of the machines are used for training while the other 20% for validation and testing purposes. The experiments show the capacity of the model to predict the last event of a given sequence only. Although no detection of attacks or malicious sequences are included in this work, interesting results are presented with respect to the behavior of LSTM models, which is one of the main objectives of this paper.

TABLE 2
Description of the Different Types of Events

ID	Event type	Actor	Target	No. Actions
0	Session	User	N/A	3
1	Process	Process	Process	5
2	Module	Process	Module (e.g. dll files)	3
3	File	Process	File	12
4	Directory	Process	Directory	14
5	Registry key	Process	Windows registry key	7
6	Registry value	Process	Windows registry value	4
7	Host Network	Process	IP address	3

Table 1 presents a summary of the focus and details found in the prior work discussed above [33], [34], [35], [36] for comparison purposes with our research. LADOHD is single layer LSTM-based anomaly detection mechanism trained with benign data only particularly applied to learn the behavior profile of an indicated application.

4 Design

Figure 2 shows LADOHD and its workflow for the detection of anomalies. The framework involves four components. First, a data generation phase, in which N machines running an EDR agent generate activity event sequences $s_1, s_2, ..., s_N$ that are collected in a centralized database. Second, a data selection step that extracts from the collected sequences the events related to the application of interest and form the subsequences $(s_1^*, s_2^*, ..., s_M^*)$. Third, a model generation phase that uses the selected subsequences to form the training and validation datasets used to train the LSTM-based model. Finally, the anomaly detector component that deploys the trained model to determine whether the events of a given testing sequence are anomalous.

4.1 Data Generation

A machine M_i runs an enterprise EDR agent that records activity events as they occur in the system. These events form a corresponding event sequence referred to as s_i. A group of

N monitored machines generate the set of event sequences $S = \{s_1, s_2, s_N\}$, which is pre-processed and used as time-series data to train the final LSTM-based model.

An activity event e_i in the sequences can be thought as a m-dimensional vector of features $\{f_i^1, f_i^2, ..., f_i^m\}$, where f_i^j represents a categorical or continuous piece of information of the reported activity. One of these features is *event type*. The EDR product generates eight event types and each has a specific set of features (including the event type itself). Namely, the number of features m varies per event type. Some features such as *event type*, *actor*, *action* and *target* are common in all the activity events (regardless their types) as they define a complete semantic for any given event e_i: "this is an event of *this type* in which *this actor* executes *this action* over *this target*." Table 2 summarizes the different types of events generated by the EDR software and the features actor, target, and action related to each type. There is a set of specific actions available to each event type. The complete list is not included per the request of the company owning the security product. An example of an event e_i and its interpretation considering the four common features listed above is as follows. The process *svchost* is an integral part of Windows OS that manages system services running from Dynamic Link Libraries (DLL). Its purpose is to speed up the startup process by loading the required services specified in the service portion of the registry. When a DLL file is loaded by *svchost*, an event of type Module is generated. The actor of the generated event is *svchost*, while *load* (predefined in the product) is the action taken over the target *DLL file*.

4.2 Data Selection

LADOHD requires the definition of a finite set of categorical events (or symbols) E, which represents the set of all possible system activity events analyzed by the LSTM-based model. This set is referred to as the vocabulary of events.

Vocabulary of Events Definition. It can be thought as a transformation function $F_T(\cdot)$ that changes the event feature vector generated by the EDR agent. Given an event $e_i = \{f_i^1, f_i^2, ..., f_i^m\}$ of type $f_i^t = f_i^{j \in \{1,2,...,m\}}$ and a set F of $k \leq m$ selected features for events of type f_i^t, the feature transformation is given by:

$$F_T(e_i, F) = \begin{cases} e_i^* = \{t_i^1, t_i^2, ..., t_i^k\} & \text{if } F \subseteq e_i \\ \varnothing & \text{otherwise} \end{cases} \quad (2)$$

In Equation 2, e_i^* is the transformed version of the event e_i. Each transformed feature $t_i^{j \in \{1,2,...,k\}}$ correspond to one of the k selected features in F. The transformation of each feature is a design choice that controls the granularity and the total number of possible events (i.e., vocabulary size). This is illustrated in Table 3 with the feature target, whose final value can be of either low or high granularity. Rows 1 and 2 are two Module events in which the same process loads two different DLL files. Assuming f_i^t ($t \in \{1, 2, ..., k\}$) were the target-related feature of these Module events, f_i^t might correspond to either the frequency of the DLL file in the distribution observed during training (low granularity) or the individual file itself (high granularity). In the former case, the two Module events would be represented by the same transformed feature vector, which translates to the same symbol in the final categorical vocabulary of events. In the latter case, two different symbols are produced. Similarly, if f_i^t were the target feature of the Host Network events in rows 3 and 4, f_i^t might either indicate whether the network connection is internal or external (low granularity) or the individual destination IP addresses (high granularity). With the low granularity interpretation, the Host Network events pass from including the entire set of IP addresses to including a binary piece of information, reducing the number of symbols that form the vocabulary.

Figure 2 shows the effect of applying the definition of the vocabulary to the selection of events. From the N original sequences, $M \leq N$ are chosen for the training phase. This is because $F_T(\cdot)$ does not produce an output when the processed event does not include the features defined in F. For a given sequence of activity events s_i, $F_T(\cdot)$ and F operate as a filter to select which events are kept and what pieces of information from them are used to generate the transformed events that form the training subsequences s_i^*. For instance, in order to build the profile of an application A, the actor feature is included in the set F and $F_T(\cdot)$ is defined so that only events with application A as actor are selected. Thereby, any sequence s_i with events produced by different applications is reduced to the subsequence s_i^*, which only includes events whose actor is application A. Any sequence s_i with no event with application A as actor is disregarded.

Based on the definition of $F_T(\cdot)$ and the selected features F for each event type, there is finite set of transformed feature vectors, which are translated one-to-one to the set of categorical symbols $E = \{1, 2, ..., |E|\}$. Whereby, a final subsequence s_i^* is comprised of these categorical symbols.

4.3 Model Generation

The selected M subsequences s_i^* are used to train the LSTM-based model following an either by-machine or by-time split. Splitting by machine refers to select approximately 80% of the the entire training subsequences s_i^* for training, leaving 20% for validation. Splitting by time, in contrast, refers to approximately select the first 80% of events in each subsequence s_i^* for training, leaving the remaining 20% of events for validation. In either splitting strategy, the resulting training and validation subsequences are concatenated to respectively form the unique training and validation sequence s_t and s_v, such that $s_t \cap s_v = \emptyset$.

Our LSTM-based model consists of a encoder of three layers of LSTM followed by a linear layer as suggested in [42]. At training, we use a timestep window $w = 1$ to compute the probability of each event of the sequence given the previous subsequence. For better results, we apply a variety of strategies such as Stochastic Gradient Descent with Restart (SGDR) [43] and Cyclical Learning Rates [44]. The hyperparamenters of the model were tuned to get the best performance for the dataset described in Section 5: (1) a batch size of 64, (2) an unrolling window (Batch Propagation Through Time or BPTT) of 64, (3) an embedding size of 16, and (4) 100 activations in the linear layer.

4.4 Anomaly Detector

At testing time, our trained LSTM-based model is used to classify each event in a sequence as either benign or

TABLE 3
Examples of Activity Events With Different Granularities

Event type	Actor	Action	Target (high granularity)	Target (low granularity)
Module	Process A	Load	DLL file1	Range 2 (100 $\leq$ frequency of file1 $\leq$ 500)
Module	Process A	Load	DLL file30	Range 2 (100 $\leq$ frequency of file30 $\leq$ 500)
Host Network	Process A	Connect	200.12.12.10	External connection
Host Network	Process A	Connect	192.168.10.3	Internal connection

anomalous. To classify the event e_t observed a timestep t, our detector follows four steps. In step 1, the previous subsequence $e_1, e_2, ..., e_{t-1}$ observed until time $t-1$ is passed as input to our trained LSTM-based model. In step 2, the model computes the probabilities of each event in E (vocabulary of events) to be the next event in the sequence given the previous subsequence. Step 3 is a procedure that creates a set of probable events $K \subset E$, whose elements are the events with the hightest probabilities. The size of the set K can be set either statically or dynamically. For the static assignment, we customize the parameter k to chose all the events in the output model whose probabilities are within the top-k probabilities. This resemble the use a fixed threshold that takes all the events above the smallest probability among the the top-k ones. The dynamic assignment, in contrast, select the most probable events based on the natural division between high and low values found in the model output. The natural division is achieved by using the most repeated probability in the output array as threshold. Probabilities above this threshold belong to the high-value set, while the remaining probabilities are assigned to the set of low values. In the final step (step 4), the event e_t is classified as benign if $e_t \in K$, otherwise anomalous.

5 Dataset

The sequences for training and testing were collected on normal and under attack conditions respectively. Security experts (referred to as Red Team) conducted specific attacks on a Windows machine during the collection period of the testing sequence. The undue activities are reflected as either specific unseen events or subsequences of expected events in an unexpected order. Table 4 summarizes the entire dataset. Pre-filter refers to the total number of events collected from multiple endpoints before filtering them out using the definition of the vocabulary of events. In contrast, post-filter refers to the sequences obtained after filtering. The collected data was filtered out to include events generated by the process *PowerShell* as actor. This is an application commonly used to conduct stealthy data exfiltration. Our intuition was that learning its behavior in normal (or non-attack) conditions would allow detecting malicious patterns resulting from the activities of the insider threat.

Vocabulary of Events. The objective was to capture as much information as possible from the PowerShell application. The selection of the set of features F of each event type was done based on the data observed in the training sequence. We traded-off granularity with the total number of possible events in order to avoid having a large number of events observed only at training (and not at testing) and vice versa. Following this guidance, each event e_i was processed using the set of features $F = \{f_i^1, f_i^2, f_i^3, f_i^4, f_i^5, f_i^6\}$, where:

TABLE 4
Dataset Description

Training / Validation			Testing	
Pre-filter	Post-filter (trn)	Post-filter (val)	Pre-filter	Post-filter (test)
38,899,995	63,282	13,280	727,275	66,972

- f_i^1: Actor feature. Its corresponding t_i^1 is a unary piece of information defining the actor (always 0 for powershell.exe).
- f_i^2: Event type feature. Its corresponding t_i^2 might be any of the eight event type IDs described in Table 2.
- f_i^3: Action feature. Its corresponding transformed featured t_i^3 was defined per event type. It can vary from 0 to 13 depending on the event type as specified in Table 2.
- f_i^4: Target feature. Its t_i^4 depends on the event type. For Process events $t_i^4 = 0$ (not powershell.exe) and $t_i^4 = 1$ (powershell.exe). For Module events $t_i^4 = 0$ (not a DLL file) and $t_i^4 = 1$ (DLL file). For Registry Value events $t_i^4 = 0$ (Others), $t_i^4 = 1$ (HKEY_USERS), and $t_i^4 = 2$ (HKEY_LOCAL_MACHINE). For the rest of event types t_i^4 operates as unitary piece of information.
- f_i^5: Network feature. Its t_i^5 is ternary piece of information about Host Network events only (0 for self connection, 1 for internal connection, and 2 for external connection). For the rest of event types this feature operates as unitary piece of information.
- f_i^6: User feature. The transformed feature t_i^6 is a binary piece of information about the user executing the action (0 for system-related user and 1 for non-system-related user). For Session and Registry Value events, this feature operates as a unary piece of information.

Each event $e_i^* = \{t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6\}$ is extracted following the definitions above. With these definitions, the vocabulary size is 175 ($E = \{0, 1, ..., 174\}$), from which 41 and 31 events are present in the training and testing sequences respectively. Twenty out of the 41 training events do not appear in the testing sequence. Likewise, ten out of the 31 testing events are not present in the training sequence. These 10 events are referred to as unseen events.

Training set. The final training (s_t) and validation (s_v) sequences were obtained by monitoring 30 machines. After filtering the collected sequences, we got a total of 76,562 events in 30 subsequences (including only events with *PowerShell* actor). We then applied a by-machine data split and selected the subsequences of the first 24 machines for training, leaving the subsequences of the remaining 6 machines

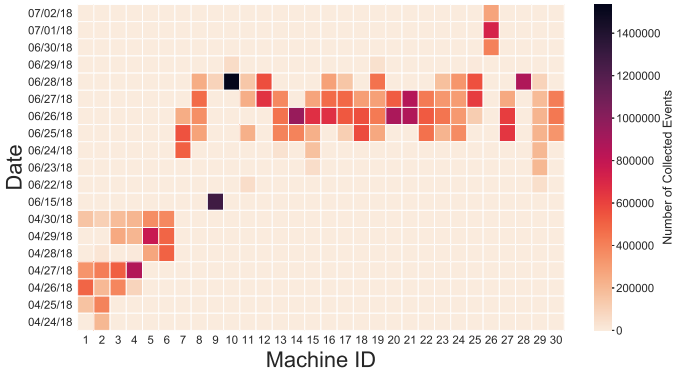


Fig. 3. Data collection dates from 30 machines, from April 27th to July 7th of 2018. This sparse collection timeframe was intended to capture the behavior of the application Powershell in non-attack conditions.

for validation. After concatenating the corresponding subsequences, the resulting training and validation sequences had a length of 63,282 (82.65%) and 13,280 (17.35%) events respectively. Figure 3 shows the collection dates from those 30 machines. This sparse collection timeframe was intended to capture the behavior of *PowerShell* in normal conditions as no attacks were reported in these collection periods.

Testing set. This sequence of 66,972 events after filtering was obtained from a different victim machine monitored from May 8th to May 11th. Among all its, a total of 117 events are unseen (observed at testing but not at training). The Red Team launched an multi-stage persistent attack consisting of a user escalation step followed by a data exfiltration attack on this victim machine as follows:

- The Red Team was initially assigned a non-administrative user for the victim computer.
- The hacking tool Mimikatz is used to steal credentials from the memory of the victim machine, performing a user escalation attack.
- The Red Team roams on the network to which the victim computer is connected and discovers other resources (shared folders, machines, users, domains). Remote shares are discovered and the corresponding files are copied to the victim machine.
- Powershell files are also copied to the machine and then executed, in order to compromise other computers and extract files from them.
- An external remote connection is established from the victim machine to bypass the firewall protection. The copied files are sent outside the network through this connection.

Ground Truth. The Red Team provided a manually generated log file enumerating the sequence of steps followed during the execution of the attack. Each entry in this file contains a high-level description of the step and a timestamp indicating the day and time of its execution. Some of the steps in the file are identifiable as actions executed by Powershell. This information was used to find the corresponding events in the testing sequence, which is formed by events in which Powershell functions as actor only. Specifically, we found 110 matches. Additionally, the EDR of the security company produced four alerts while the Red Team conducted the attack on the victim machine. These alerts were matched with 8 events in the testing sequence. The alert-related events (8) along with the Red-Team-matched events (110) and the unseen events (117) form the sequence portions corresponding to malicious activity. Namely,there are 295 malicious events throughout the testing sequence. . Malicious events are expected to be detected as anomalous. Events other than alert-related, Red-Team-matched, and unseen events are regarded benign and are expected to be as such. We set our ground truth based on these assumptions and measure the performance of our method accordingly.

6 Experiments

This section presents the experiments conducted to evaluate our LADOHD framework using the datasets described in Section 5. The experiments were designed to answer a series of of research questions, which are included along with our findings in the following sections.

6.1 Dynamic vs. Static Selection of the Set of Probable Events K

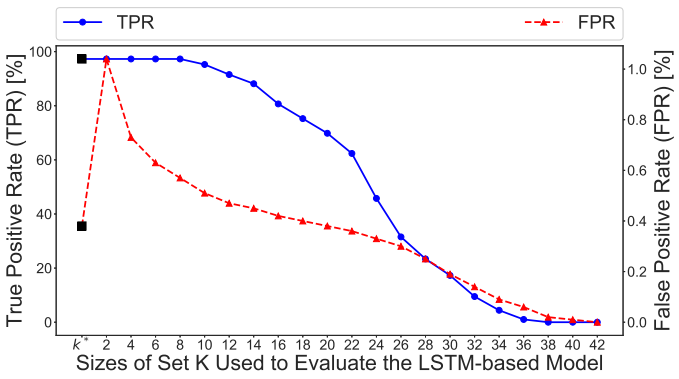
RQ1. What approach (either dynamic or static selection of K) provides a better performance? If the static method does, what is the best value of the parameter k?

We investigate whether our technique identifies the well-identified malicious events. We are particularly interested in finding which approach provides the best anomaly detection performance. To this end, we measure the TPR and FPR variations as we change the number of events in K through both the dynamic and static approaches.

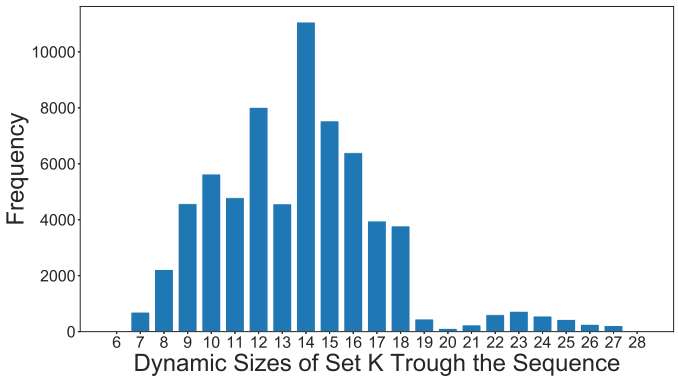
Figure 4a shows the results. In the x-axis k^* means that the size of K is dynamically adjusted in each timestep of the sequence. The corresponding values of the metrics TPR and FPR are marked with a black square to differentiate them from the values obtained through the static approach. The rest of values in the x-axis (from 2 to 42) corresponds to the values of the parameter k of the static approach. The figure illustrates how the dynamic approach outperforms the static approach for any chosen k as the former gives a high TPR of 97.29% while keeping a low FPR of 0.38%.

The static approach starts with a similar TPR but a higher FPR in comparison with the dynamic approach. As the parameter k increases, both the TPR and the FPR decrease. The static approach achieves the same FPR as the dynamic when $k = 20$. At this point however, the corresponding TPR has decreased from 97.29% to 69.83%. The non-functionality of the static method can be explained by the high variance in the distribution of the natural division between high and low values in output of the model throughout the entire sequence. Figure 4b shows this distribution. The dynamic approach produces sets K with sizes between 6 and 28 (inclusive) with significant differences in their frequencies. Setting a specific k in the static approach to be used in each timestep of the analyzed sequence goes away from the decision made by the model, which clearly discriminates low and high values in the its output probabilities.

Findings. The dynamic approach outperforms the static method to select the set of probable events K, having a 1.31X higher TPR for the same FPR of 0.38.



(a) Metric Variation With Static and Dynamic Selection of K



(b) Distribution of Dynamic K Sizes Throughout the Sequence

Fig. 4. Selection of the set K through both the dynamic and static approaches. Figure 4a shows the variation of the TPR and FPR with respect to different setting for K. Figure 4b presents the distribution of the dynamic sizes of K throughout the testing sequence. Notice its high variability.

6.2 Comparison With an Enterprise Endpoint Detection and Response (EDR)

RQ2. What is the performance of LADOHD with respect to enterprise-level EDR system currently in production?

One of the main challenges defending against insider threats is the similarity between benign and malicious activities. Discerning between them is a difficult task. Due to this restriction, this section evaluates how efficient LADOHD is by comparing it with the the enterprise EDR of the company already in production.

The enterprise EDR is a multi-layer system that employs signature-based, supervised (e.g. Random Forest), and unsupervised (e.g., clustering) machine learning methods for analysis, detection, and alerting. This system detected 4 alerts while monitored the victim machine during the Red Team attack. These alerts were validated to correspond to malicious activities and later matched with the corresponding events in the generated testing sequence as indicated in Section 5. The EDR did not detect the complete multistage persistent attack, but rather a few steps. Figure 5 shows the malicious activities reported by the Red Team in the log file that were matched with events in the testing sequence. It shows their counts in log file and the number of events matched per each activity. There is a total of 110 malicious events. All of them were classified by LADOHD as anomalous.

Unlike the current analytics modules of the EDR, LADOHD is trained with benign data only and learns sequence patters of a particular application. Uncommon event sequences are classified as anomalous through the detection of specific events regardless the meaning of the event itself. It is important to mention that LADHD produced 254 FP cases along with the with the 287 TP cases (including unseen events). Although the number of FPs is low with respect to the length of the sequence, they might represent a high volume of cases to be revised by security experts in a short periods of time.

Findings. LADOHD successfully detected the ongoing attack generating more alerts than the enterprise EDR. A relatively small number of FP cases with respect to the sequence length were generated in the process.

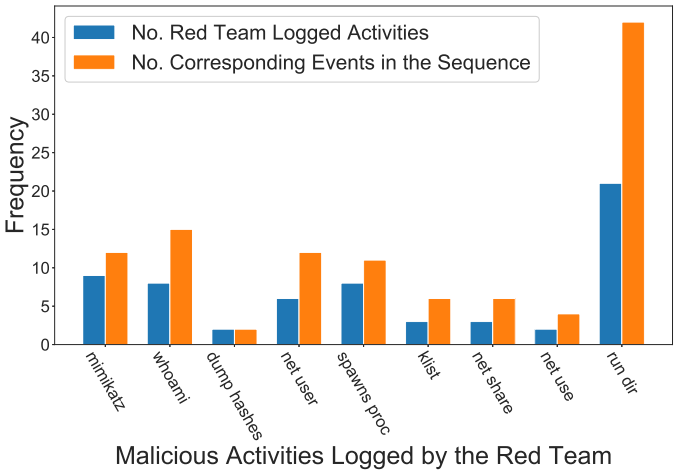


Fig. 5. Malicious activities reported by the Red Team. These are the activities that could be matched with events in the testing sequence.

6.3 Performance of LADOHD When Processing Sequences Without Unseen Events

RQ3. Does the capacity LSTM-based models to detect anomalies improve when unseen events are discarded in advance?

We now evaluate whether processing unseen events improve or diminish the capacity of LSTM-based models to detect malicious events. We are interested in knowing whether the same Red-Team-matched events classified as anomalous in experiment 1 (Section 6.1), when unseen events are part of the sequence, are also classified as anomalous when unseen events are ignored. We also want to validate whether the missing alert-related events are detected. To this end, we create a clean testing sequence by removing the unseen events from the testing sequence. We pass this clean testing sequences to our LSTM-based model, which classifies each event in the sequence as either benign or anomalous. Table 5 includes a comparison of the results obtained with the original testing sequence and its clean version. Removing the unseen events does not help the model classify as anomalous new malicious events. The number of TP cases related to observed events remains the

TABLE 5
Metric Values Obtained With the Original and Clean Testing Sequences

Testing Sequence	TP	FN	FP	TN	TPR	FPR
Original	287	8	254	66423	97.29%	0.38%
Clean	110	8	256	66421	93.22%	0.38%

same. The same phenomenon occurs with the number of FPs, where an increase of only 2 is observed.

Findings. There is neither a significant improvement nor detriment when unseen events are discarded. Their removal from the sequence should be determined by the performance cost they might cause only.

6.4 Effect of Long-Term Dependencies in the Detection of Anomalies

RQ4. What impact does the length of the previous sequences have over the detection of anomalies?

One of the main characteristics of LSTM networks is their capacity to learn long-term dependencies among events in a sequence. One interesting question we aim to answer is whether these long-term dependencies have an impact in the classification. Namely, we want to validate whether considering subsequences of different lengths (by increasing the number of predecessors) cause different outputs in the classification of an event. To this end, we work with the clean testing sequence of the Section 6.3.

Table 6 shows the results. It includes 9 randomly selected events of these sequence classified as anomalous. These events are referred to as targets and they correspond to the last events of low probable subsequences in the sequence. Figure 6 illustrates the procedure followed in this experiment using the target e_{145} (row 4 in the table) as example. We incrementally move backward from each target until the beginning of the sequence and find the number of predecessors (length of previous sequence) that causes a change in the classification. The number of predecessors that causes the classification to be benign are called benign shifters. Those that cause the classification to be anomalous are referred to as anomalous shifters. Table 6 shows that most of the targets have multiple shifters, which prove that the history of events is what actually has a significant impact in the classification process. LSTM-based models have the potential to correctly classify events regardless the length of the previous sequence. Their outputs are in fact affected by the hidden state. An interesting observation is the capacity of our LSTM-based model to look backward a variable number steps to estimate the probability of a particular event. We have cases where the model makes the final decision based on a few number of predecessors (e.g., 2 predecessors in the case of e_{142}) and others in which the model considers a large number of them (e.g., 648 predecessors in the case of e_{1032}). This ability to relate current events with distant past events in the sequence shows the potential of LSTM networks to solve the OAR problem.

Another question we aim to answer in this experiment is how the probabilities of the targets change as the number of predecessors gets close to a shifter. We did not find a clear

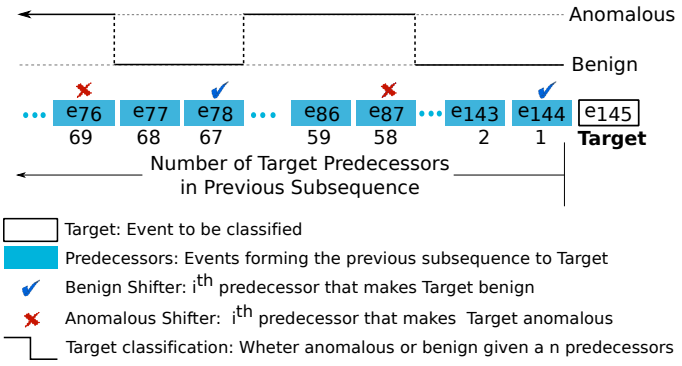


Fig. 6. Effect of long-term dependencies of LSTM models in the detection of anomalies. The example is based on the event e_{145} in Table 6

TABLE 6
Effect of Long-Term Dependencies on the Detection of Anomalies

Target	Benign Shifters	Anomalous Shifters
e_{142}	[1]	[98]
e_{143}	[1]	[75]
e_{144}	[1]	[75]
e_{145}	[1, 67]	[58, 69]
e_{146}	[1]	[54]
e_{147}	[1]	[50]
e_{148}	[1]	[47]
e_{1032}	[1, 551, 619]	[549, 587, 648]
e_{2292}	[9]	[1, 11]

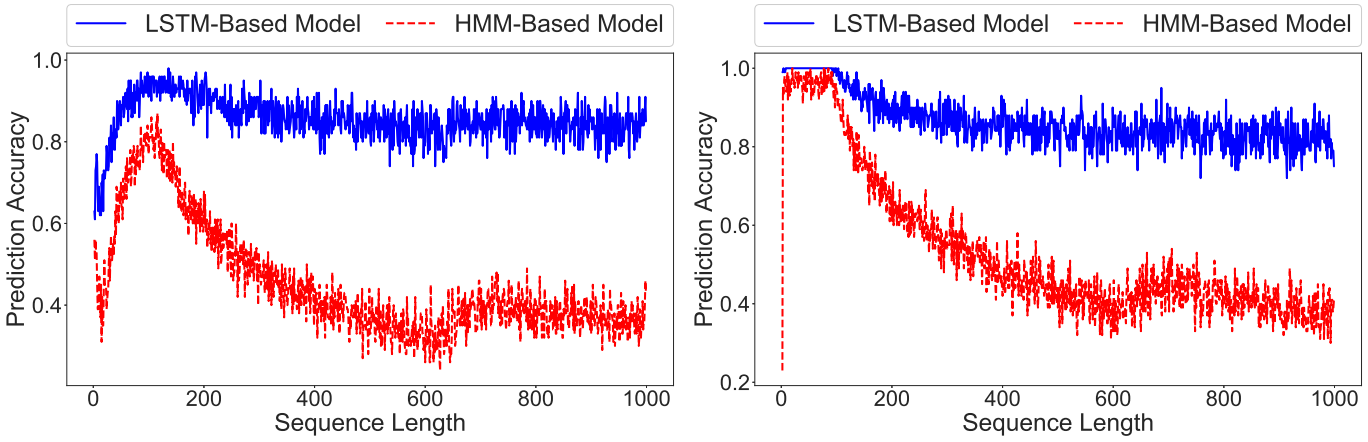
relationship between the relative probability of the target (with respect to the other possible events) in the output of the model and the proximity to the shifters. The probability of the target does not always increase or decrease as the number of predecessors gets close to a benign or anomalous shifter respectively. This phenomenon can be explained by the fact that our model is trained to predict the next event in the sequence and not to estimate the least probable events.

Findings. LADOHD can correctly classify events regardless the length of the previous subsequences. The history of events (hidden state of the model) is what actually affects the classification. The ability to look backward more than 600 steps (e.g. target e_{1032}) show the potential of this solution against the OAR problem.

6.5 Prediction Capacity of LSTM and HMM Based models Over Variable-Length Sequences

RQ5. How much accurate are LSTM-based models compared to other solutions such as HMM in the prediction of the next event in variable length sequences?

This section presents a comparison of the prediction capacity of our LSTM-based model and a full connected HMM model built with the same dataset. As the ability to discern between benign and anomalous events depends on the prediction capacity of the model, we want to evaluate which model predicts better the last event of sequences of different lengths. To this end, we took 100 continuous subsequences of a specific length and measure the accuracy of the model predicting the last event of the sequences. The lengths were chosen to vary from 2 to 1000. To ensure a fair



(a) Accuracy Witn Training Sequences

(b) Accuracy With Testing Sequences

Fig. 7. Prediction accuracy of our LSTM and the HMM models with sequences of incremental lengths. Figure 7a and Figure 7a show the accuracy variation with subsequences extracted from the D1 training and testing sequences respectively.

comparison, we measure the prediction accuracy of both models with incremental-length subsequences extracted from both the training and testing sequences. The idea of using these two sets of subsequences is to validate that the results do not come from any bias that the testing data might induce. We do so because we are interested in observing how the prediction capacity of the models changes as the lengths of the sequences increase in ideal conditions (i.e. when sequences were observed in training) and not in comparing which model is more efficient when processing new data. Figure 7 shows that our LSTM-based model constantly outperform the prediction capacity of the HMM model. Our model keeps predicting well with larger sequences, while the accuracy of the HMM-based model decreases.

Findings. LSTM-based models have a better capacity than HMM-based models to predict the next event in a given sequence as its length increases.

7 Conclusion

This paper presents LADOHD, a generic LSTM-based anomaly detection framework to protect against insider threats. for high dimensional sequential data. We evaluated the framework with an extensive dataset of activity events generated by the EDR of a renown security company. Each event in the dataset represents a hight dimensional vector of features The framework filters out the events per application and pre-specified features that define the vocabulary of possible events that form the sequences analyzed by our model. Each event in the sequence is classified as either benign or anomalous given the previous observed subsequence. LADOHD reached a high $TPR > 97\%$ with a low $FPR < 0.4\%$,proving the effectiveness of the framework. Furthermore, this work presents a comprehensive analysis of how LSTM-based models work and compare them to alternative solution such as HMM-based models. We found that LSTM-based models rank better the set of expected events in each timestep of sequence than HMM-based models, which favor their capacity to detect anomalies.

References

[1] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security*. Springer, 2008, pp. 69–90.

[2] A. Sanzgiri and D. Dasgupta, "Classification of insider threat detection techniques," in *Proceedings of the 11th annual cyber and information security research conference*, 2016, pp. 1–4.

[3] J. Hunker and C. W. Probst, "Insiders and insider threats-an overview of definitions and mitigation techniques." *JoWUA*, vol. 2, no. 1, pp. 4–27, 2011.

[4] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *IFIP International Conference on Communications and Multimedia Security*. Springer, 2014, pp. 63–72.

[5] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 255–264.

[6] H. Xu, W. Du, and S. J. Chapin, "Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 21–38.

[7] J. T. Giffin, S. Jha, and B. P. Miller, "Automated discovery of mimicry attacks," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 41–60.

[8] D. Yao, X. Shu, L. Cheng, and S. J. Stolfo, "Anomaly detection as a service: Challenges, advances, and opportunities," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 9, no. 3, pp. 1–173, 2017.

[9] K. Xu, D. D. Yao, B. G. Ryder, and K. Tian, "Probabilistic program modeling for high-precision anomaly classification," in *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*. IEEE, 2015, pp. 497–511.

[10] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[11] M. Agyemang, K. Barker, and R. Alhajj, "A comprehensive survey of numeric and symbolic outlier mining techniques," *Intelligent Data Analysis*, vol. 10, no. 6, pp. 521–538, 2006.

[12] Z. A. Bakar, R. Mohemad, A. Ahmad, and M. M. Deris, "A comparative study for outlier detection techniques in data mining," in *2006 IEEE conference on cybernetics and intelligent systems*. IEEE, 2006, pp. 1–6.

[13] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005, vol. 589.

[14] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[15] P. Thompson, "Weak models for insider threat detection," in *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III*, vol. 5403. International Society for Optics and Photonics, 2004, pp. 40–48.

[16] M. B. Salem and S. J. Stolfo, "Masquerade attack detection using a search-behavior modeling approach," *Columbia University, Computer Science Department, Technical Report CUCS-027-09*, 2009.

[17] C. Warrender, S. Forrest, and B. A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," 1999.

[18] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.

[19] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.

[20] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[21] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern recognition*, vol. 36, no. 1, pp. 229–243, 2003.

[22] K. Xu, K. Tian, D. Yao, and B. G. Ryder, "A sharper sense of self: Probabilistic reasoning of program behaviors for anomaly detection with context sensitivity," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 467–478.

[23] J. Hollmén and V. Tresp, "Call-based fraud detection in mobile communication networks using a hierarchical regime-switching model," in *Advances in Neural Information Processing Systems*, 1999, pp. 889–895.

[24] P. Smyth, "Clustering sequences with hidden markov models," in *Advances in neural information processing systems*, 1997, pp. 648–654.

[25] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Visualization of navigation patterns on a web site using model-based clustering," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 280–284.

[26] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[27] G. Salton, R. Ross, and J. Kelleher, "Attentive language models," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2017, pp. 441–450.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] M. Villarreal-Vasquez, *LADOHD Repository to be Public After Revision*, 2020. [Online]. Available: https://github.com/mvillarreal14/ladohd

[31] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[32] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, 2015, pp. 2773–2781.

[33] G. Kim, H. Yi, J. Lee, Y. Paek, and S. Yoon, "Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems," *arXiv preprint arXiv:1611.01726*, 2016.

[34] C. Feng, T. Li, and D. Chana, "Multi-level anomaly detection in industrial control systems via package signatures and lstm networks," in *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 2017, pp. 261–272.

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

[36] Y. Shen, E. Mariconti, P.-A. Vervier, and G. Stringhini, "Tiresias: Predicting security events through deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018.

[37] K. Rocki, "Recurrent memory array structures," 07 2016.

[38] G. Bouchard, "Efficient bounds for the softmax function, applications to inference in hybrid models," in *Presentation at the Workshop for Approximate Bayesian Inference in Continuous/Hybrid Systems at NIPS-07*. Citeseer, 2007.

[39] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[41] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[42] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[43] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[44] L. N. Smith, "Cyclical learning rates for training neural networks," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 464–472.

Miguel Villarreal-Vasquez Miguel Villarreal-Vasquez is a Ph.D. Candidate in computer science at Purdue University, West Lafayette, IN 47907, USA. He a research and teaching assistant in the same institution His research interests are building resilient security systems using machine learning, with special attention to deep learning. His research also embraces improving the resiliency of deep learning architectures against adversarial settings. In 2018, he worked for the Center for Advanced Machine Learning (CAML) at Symantec, Mountain View CA, where he started this research work.

Gaspar Model-Howard Gaspar Modelo-Howard (SM'12) received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN 47907, USA, in 2013. He is a principal data scientist for the Public Cloud Security team at Palo Alto Networks, Santa Clara, CA 95054 USA. His research interests are network and cloud security, with a focus on web security, intrusion detection and response, and malware detection. Dr. Modelo-Howard is an adjunct professor in computer security at Universidad Tecnológica de Panamá and also member of ACM and Usenix.

Bharat Bhargava Bharat Bhargava (F'90) received the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN 47907, USA, in 1974. He is a professor of computer science at Purdue University and is leading a Northrop Grumman Corporation (NGC) sponsored consortium on Real Applications of Machine Learning (REALM) with MIT, CMU and Stanford. He is contributing to Department of Defense on The Science of Artificial Intelligence and Learning for Open-world Novelty (SAIL-ON) and another project with NGC on explainable AI and adversarial machine learning. Prof. Bhargava is the founder of the IEEE Symposium on Reliable and Distributed Systems, IEEE conference on Digital Library, and the ACM Conference on Information and Knowledge Management. He has won eight best paper awards in addition to the technical achievement award and golden core award from IEEE. He received Outstanding Instructor Awards from the Purdue chapter of the ACM in 1996 and 1998. In 2003, he was inducted in the Purdue's Book of Great Teachers.

Simant Dube Simant Dube received the Ph.D. degree in computer science from University of South Carolina, Columbia, SC 29208 USA, in 1992. He is Technical Director in the Center for Advanced Machine Learning (CAML) at Symantec, Mountain View, CA 94043 USA. His areas of interest are artificial intelligence, machine learning, cybersecurity and computer vision. He has worked in companies such as Fluidigm, Microsoft and Amazon on diverse applications of AI and machine learning. Dr. Dube held the Ramanujan Fellowship at Indian Institute of Technology, Kanpur.