Hunting for Insider Threats Using LSTM-Based Anomaly Detection

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

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 outperforms the anomaly detection system used to protect the commercial network with a True Positive Rate of 97.29% and a False Positive Rate of 0.38%. Experiments also show that LSTMs have higher prediction precision in variable-length sequences than methods like Hidden Markov Models, a crucial requirement in sequence-analysis-based anomaly detection techniques.

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 attacks [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

- *Miguel Villarreal-Vasquez and Bharat Bhargava are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA. E-mail: {mvillar, bbshail}@purdue.edu.*
- *Gaspar Modelo-Howard is with Palo Alto Networks, Santa Clara, CA 95054 USA. E-mail: gaspar@acm.org.*
- *Simant Dube is with Broadcom Inc., Mountain View, CA 94043 USA. E-mail: simant.dube@broadcom.com.*

Manuscript received 23 June 2020; revised 2 Sept. 2021; accepted 4 Nov. 2021. Date of publication 15 Dec. 2021; date of current version 16 Jan. 2023.
This work was supported in part by Northrop Grumman Cybersecurity Research Consortium (NGCRC).
(Corresponding author: Miguel Villarreal-Vasquez.)
Digital Object Identifier no. 10.1109/TDSC.2021.3135639

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 that they are incapable to discern the order of events in long sequences due to the OAR problem, restricting the 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 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.
- 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 detection rate above 97% while keeping a FPR $< 0.4\%$. We study the feasibility of incorporating LADOHD as a complement module to the EDR system as our framework detected more malicious events under the same attack.
- A deep analysis of the features of the events generated by the EDR is presented. Features were selected to form a vocabulary of events that allow the model to successfully learn 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 other sequence-based methods (e.g., HMM-based methods) to solve the OAR problem.

2 Overview and Threat Model

2.1 Overview

Strategy. 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].

Vocabulary of Events. LADOHD requires the definition of a finite set of possible symbols $E = \{1, 2, \ldots, N\}$, 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 $S = \{s_1, s_2, \ldots s_N\}$ 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.

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

Evaluation. During the evaluation phase, given a previous sequence of events until timestep $t-1$ $e_1, e_2, \ldots, e_{t-1}$ $(e_i \in E)$, 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 $s = e_1 e_2 \ldots e_{t-1}$, our framework computes the probabilities of possible events next in the sequence $P(e_i|e_{1:i-1})$ for $i = 1, 2, \ldots$

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 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 an 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 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.

3.2 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 take 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.

3.3 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 that 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 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 shown a significant improvement in the TPR. All the described HMM based methods [9], [21], [22] applied the dynamic programming algorithm Viterbi [39] for inference. The time complexity of this algorithm is $O(|S|^2)$, with S being the set of hidden states [40]. 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.

3.4 Anomaly Detection Based on Sequence Analysis Using LSTM

Some research work have endeavored to investigate the application of LSTM-based models to anomaly detection and similar security problems [33], [34], [35], [36]. In

TABLE 1
Comparison With Existing LSTM-Based Security Solutions

Research	Strategy	Hunts Insider Threats	LSTM-Only-Based Architecture	Basic Analysis	Extended Analysis
System Call Language Modeling [33]	Intrusion detection	✗	✗	✗	✗
Multi-level Detector (For ICS) [34]	Anomaly detection	✗	✗	✗	✗
Deeplog [35]	Anomaly detection	✗	✓	✗	✗
Tiresias [36]	Attack step prediction	✗	✓	✓	✗
Insider Threat Detection with DNN [37]	Intrusion detection	✓	✗	✓	✗
Insider Threat Detection with LSTM [38]	Anomaly detection	✓	✓	✓	✗
LADOHD [this work]	Anomaly detection	✓	✓	✗	✓

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. 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 events. An event is considered anomalous if its probability is not among the top-*k* output probabilities of the model. 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. The LSTM model is trained using a small portion of non-malicious data to determine the next action to occur given a previous sequence of actions. For each identified action, a different LSTM model is trained with the goal of not only determining the expected action 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. A sequence is considered anomalous if there exists at least one action whose probability is not among the highest top-*k* from the model. The limits of LSTM models with respect to the length of the sequences is not evaluated in this work.

A more recent work, called TIRESIAS [36], uses LSTM-based models to predict the next step in an already detected attack given the previous sequence of steps in such attack. The sequences generated by 80% of the machines observed are used to train the model and learn the order of events of the detected attacks. No detection of attacks or malicious sequences are included in this work, but interesting results are presented with respect to the behavior of LSTM models.

Yuan *et al.* [37] present an intrusion detection framework for insider threats. The framework is based on a LSTM-CNN architecture trained in a supervised manner using the CERT dataset [41]. The technique creates user profiles from activity sequences collected in a daily basis. Each profile uses a LSTM and a CNN model trained with label data. In addition to the limitation imposed by the requirement of labels, this work was conducted with a reduced vocabulary comprised of only 16 possible events.

Lu *et al.* [38] extend the previous work and present a LSTM-based anomaly detection technique also tested over the CERT dataset [41]. The method profiles users with a vocabulary of 187 possible events. The work proposes two hyperparamenters. First, a tolerance factor g, which functions as a fixed threshold used to claim as anomalous any event that is not among the first g-top possible events at a given timestep of the sequence. The second hyperparameter refers to the block element size n. It indicates how many previous events are taken into consideration to predict the next event. The presented experiments include results for $1 \leq n \leq 10$. Besides the shortcoming caused by this small size of n, this work does not include an analysis of the strengths and limitations of LSTM architectures on solving the anomaly detection problem.

Table 1 presents a summary of the focus and details found in the prior work discussed above [33], [34], [35], [36], [37], [38] for comparison purposes with our research. LADOHD is a LSTM-based anomaly detection mechanism trained with benign data only applied to learn the behavior profile of an indicated application.

4 Design

Fig. 1 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 $s_1, s_2, \ldots, s_N$ 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 $(s_1^*, s_2^*, \ldots, s_M^*)$. 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 sequential data to train the final LSTM-based model.

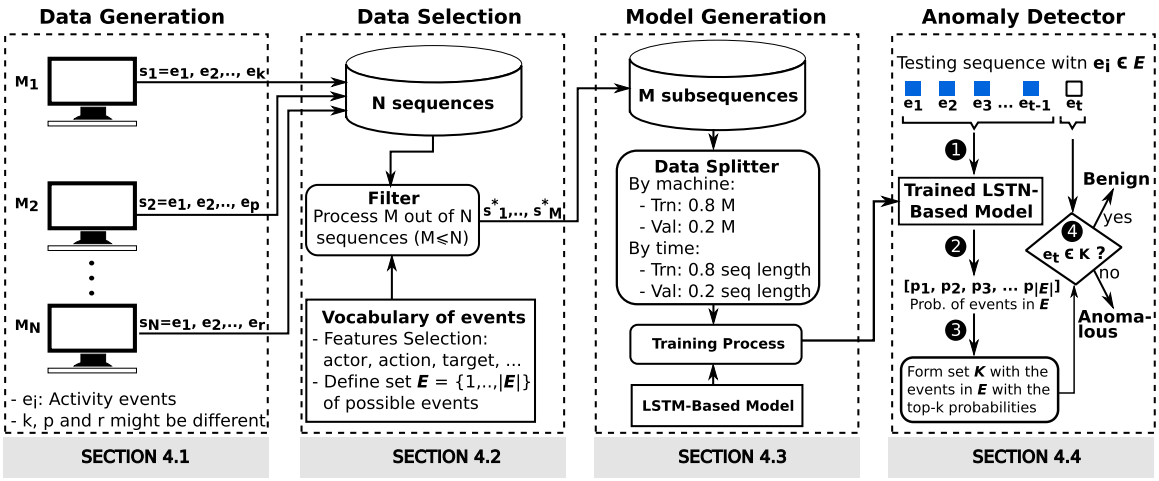


Fig. 1. Components of our anomaly detection framework LADOHD to counter insider threats: (1) data generation, (2) data selection, (3) model generation, and (4) anomaly detector. Below each component, there is a reference to the section providing a detailed explanation about its operation.

An activity event e_i in the sequences can be thought as a m-dimensional vector of features $\{f_i^1, f_i^2, \ldots, f_i^m\}$, 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* 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 $e_i = \{f_i^1, f_i^2, \ldots, f_i^m\}$ of type $f_i^t = f_i^{j\in\{1,2,\ldots,m\}}$ and a set F of $k \leq m$ selected features for events of type f_i^t, the feature transformation is given by:

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

In Equation (1), e_i^* is the transformed version of the event e_i. Each transformed feature $t_i^{j\in\{1,2,\ldots,k\}}$ 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 $(t \in \{1, 2, \ldots, k\})$ were the target-related features of these Module events, f_i^t might correspond to either the frequency of the DLL files in the distribution observed during training (low granularity) or the individual files themselves (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 features 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.

Fig. 1 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

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 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 ≤ 500)
Module	Process A	Load	DLL file30	Range 2 ($100 \leq$ frequency of file30 ≤ 500)
Host Network	Process A	Connect	200.12.12.10	External connection
Host Network	Process A	Connect	192.168.10.3	Internal connection

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 it is used to generate the transformed events that form the training subsequence 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 $E = \{1, 2, \ldots, |E|\}$. Whereby, a final subsequence s_i^* is comprised of these categorical symbols.

4.3 Model Generation

The selected M subsequences $\{s_1^*, s_2^*, ..., s_m^*\}$ 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 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 sequences 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 anomalous. To classify the event e_t observed a timestep t, our detector follows four steps. In step 1, the previous subsequence $e_1, e_2, \ldots, e_{t-1}$ 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 of 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 and Evaluation

Dataset. The sequences for training and testing were collected on normal and under attack conditions respectively on different Microsoft Windows machines and different times. The final training (s_t) and validation (s_v) sequences were obtained by monitoring 30 machines, which were isolated and operated in non-attack conditions. These machines generated 38.9 million benign events that were collected in the dates indicated in Fig. 2. The sparse collection timeframe was intended to capture the behavior of different actors in normal conditions as no attacks were reported in these collection periods. The collected benign events were filtered out using the definition of the vocabulary of events to generate the final sequences s_t and s_v of a selected group of processes (i.e., actors). These sequences were then used to create the corresponding application profiles. We profiled 6 out of 695 applications present in the benign data. We chose them based on their popularity as attack vectors in Microsoft Windows systems as found in collections of adversarial techniques used by the cybersecurity industry, such as the MITRE ATT&CK framework [45]. A description of the profiled processes and their capacity to be exploited in an insider threat attack are listed below.

- *cmd.exe*: Starts a new instance of the command interpreter, which is used by adversaries to execute commands and payloads.
- *cscript.exe*: May run scripts signed with trusted certificates to proxy the execution of malicious files.
- *net.exe*: Leveraged by attackers to gather system and network information or to interact with services.
- *powershell.exe*: Powerfull scripting environment abused by adversaries to execute scripts, modify other processes, and perform lateral movements.

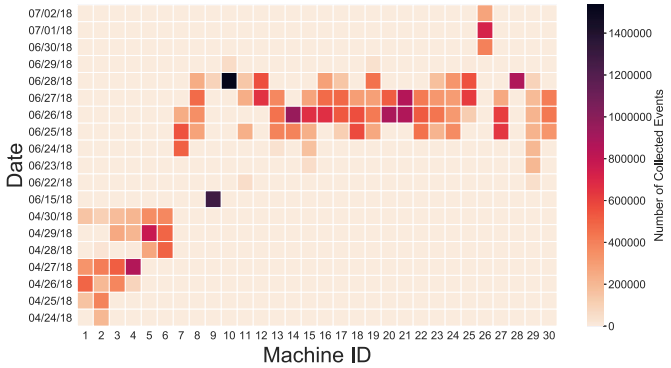


Fig. 2. Data collection dates from 30 machines, from April 27th to July 7th of 2018. This sparse collection timeframe was intended to capture the behavior of a variety of applications in non-attack conditions.

TABLE 4
Training and Validation Data[1] to Profile the Selected Actors and Their Corresponding Testing Data[2]

Actors	RD/BD Ratio	Benign Data (BD)		Red Team Data (RD)
		Training	Validation	Testing
svchost.exe	2.88%	9,141,431	2,077,802	323,296
cscript.exe	0.00%	995,053	226,171	0
xcopy.exe	0.00%	394,587	89,688	0
cmd.exe	5.04%	198,349	45,084	12,257
powershell.exe	87.47%	63,282	13,280	66,972
net.exe	0.93%	22,148	5,035	254
No. of Events		**10,814,850**	**2,457,060**	**402,779**

[1]*From the 38,899,995 events collected from 30 machines.*
[2]*From the 727,275 events collected from the victim machine.*

- *svchost.exe*: Runs DLL-based services. DLLs may be injected into processes to either evade process-based defenses or escalate privileges.
- *xcopy.exe*: Copies files and directories, which can be used in data exfiltration attacks.

The final testing sequence was obtained by monitoring a different Windows machine from May 8th to May 11th, 2018. Security experts, to which hereinafter we will refer to as Red Team, conducted a multi-step attack on the victim machine during the collection period of the testing sequence. A total of 727,275 events were generated by the victim machine, including 162 different actors. We filtered out these events using the same definition of vocabulary of events applied to the training data for the same 6 actors. Our intuition was that malicious activities executed during the attack would be reflected as either specific unseen events or subsequences of expected events in an unexpected order in the collected testing sequence. Table 4 summarizes the training, validation, and testing data used in our experiments.

Attack. 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:

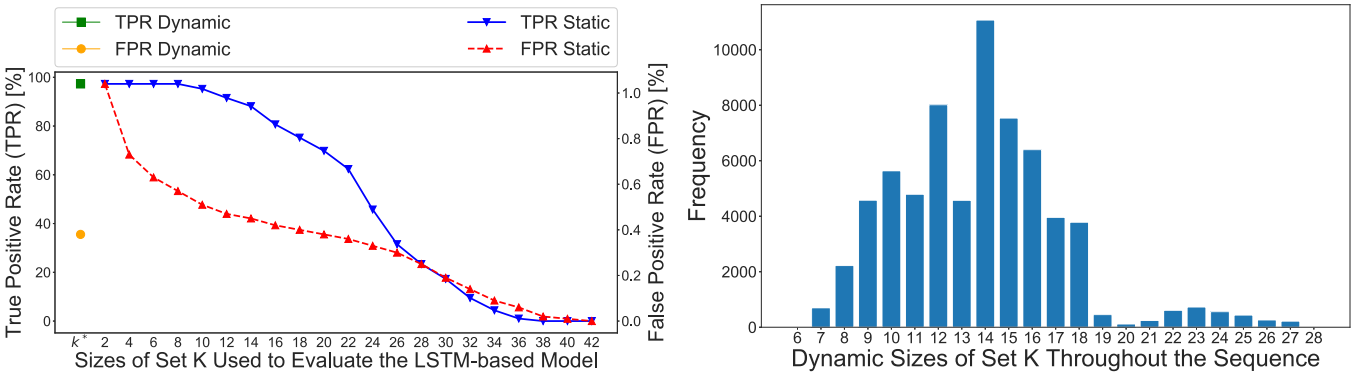
- The Red Team was initially assigned a non-administrative user for the victim computer.
- The hacking tool Mimikatz is used to steal credentials from the memory of the victim machine, performing a user escalation attack.
- The Red Team roams on the network to which the victim computer is connected and 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 and send the copied files outside the network.

Considering the number of events at testing to training ratio (RD/BD) of each actor shown in Table 4, it is noticeable that the Red Team conducted most of their activities through Powershell. This behavior is not unusual in insider threat attacks as described in [46]. Despite this observation, we looked for anomalies in the testing sequences of svchost.exe, cmd.exe, powershell.exe, and net.exe through LADOHD. We only found anomalies when analyzed powershell.exe. Whereby, we will develop the rest of this paper focusing on the details and experiments performed with this actor.

Vocabulary of Events. The objective was to capture as much information as possible from the powershell.exe 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:

- 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, f_i^6 is 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 ($E = \{0, 1, \ldots, 174\}$), 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



(a) Metrics with the Dynamic and Static Methods

(b) Distribution of Dynamic K Sizes Throughout the Sequence

Fig. 3. Performance of the dynamic and static methods. Fig. 3a shows the TPR and FPR obtained with a dynamic K (K^*) and their variations with varying static values of it. Fig. 3b shows the distribution of the dynamic sizes of K throughout the testing sequence. Notice its high variability.

events are not present in the training sequence. These 10 events are referred to as unseen events.

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 a step and a timestamp indicating the day and time of its execution. Some of the steps in the file are identifiable as actions executed with Powershell. This information was used to find the corresponding anomalous events in the testing sequence. Specifically, we identified 295 malicious events comprised of 118 entries in the log file and 177 unseen events at training.

6 Experiments

6.1 Dynamic Versus 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 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.

Fig. 3a shows the results. In the x-axis, k^* means that the size of K is dynamically adjusted in each timestep of the sequence. The values of the metrics TPR and FPR obtained with the dynamic approach are plotted with a green square and a orange circle respectively to differentiate them from the values obtained through the static method.

The rest of values in the x-axis (from 2 to 42) correspond to the values of the parameter k adjusted through 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 is statically increased, both the TPR and the FPR decrease. The static approach achieves a similar FPR as the dynamic when $k = 20$. At this point however, the corresponding TPR has decreased 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 the output of the model throughout the entire sequence. Fig. 3b shows this distribution. The dynamic approach produces sets K with sizes between 6 and 28 (inclusive) with significant differences in their frequencies. Setting a fixed k through the static method to be used in each timestep of the sequence goes away from the decision made by the model, which dynamically discriminates between low and high values in its output probabilities.

Findings. The dynamic approach outperforms the static method to select the set of probable events K, having a 1.39X higher TPR for the same FPR of 0.38.

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

RQ2. What is the performance of LADOHD with respect to the 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 challenge, this section evaluates how efficient LADOHD is by comparing it with enterprise EDR of the company already in production. The idea is to evaluate the feasibility of incorporating LADOHD as a complement to the monitoring stack of the EDR.

The enterprise EDR is a multi-layer system that employs signature-based engines to find threats in network traffic, along with sandboxing and machine learning (random forest and clustering) to analyze suspicious files and compare them against known, malicious files. The EDR also accesses a reputation database with over 8 billion files, public IP addresses, and domains, to validate its results. The EDR system detected 8 malicious events from the activity generated by the Red Team attack, which were reported as 4 alerts. We confirmed that all alerts were true positives.

Fig. 4 shows the counts of the malicious activities executed through Powershell reported by the Red Team in the log file and the corresponding number of events matched in the sequence. A total of 118 malicious events were reported in the log file. LADOHD classified 110 of them as anomalous. None of these detecte events correspond to the the 8 events detected by the EDR.

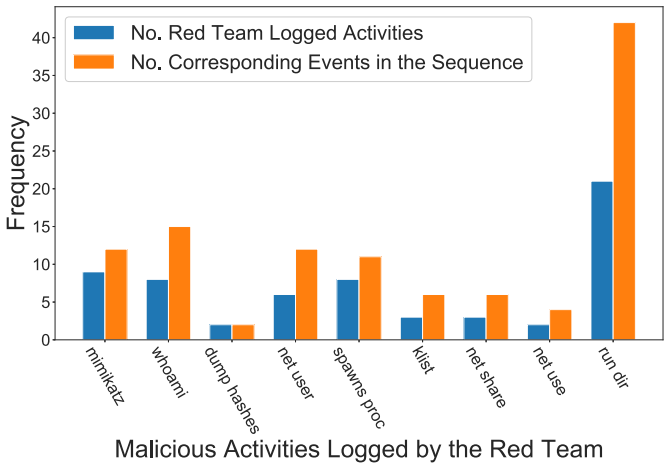


Fig. 4. Malicious activities reported by the Red Team. These are the activities that could be matched with events in the testing sequence.

TABLE 5
Anomaly Detection Comparison Between the EDR and LADOHD

Solution	Recall	Precision	F1-Score
EDR	2.79%	100.0%	0.024
LADOHD	97.29%	53.04%	0.85

Unlike the current analytics modules of the EDR, LADOHD is trained with benign data only and learns sequence patterns of a particular application. Both uncommon event sequences and unseen events are classified as anomalous regardless the meaning of the events. The testing sequence includes 177 unseen events. As expected, LADOHD classified all of them as anomalous. Table 5 summarizes the results from the significant tests run on both LADOHD and the EDR.

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.

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 previous experiments 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 8 missing events related to the missing alerts detected by the EDR can be detected by LADOHD when unseen events are removed. To this end, we create a clean testing sequence by removing the unseen events from the testing sequence. We pass this new sequence to our model, which classifies each event in it as either benign or anomalous. Table 6 includes the results obtained with the original testing sequence and its clean version. Removing the unseen events does not help the model detect new malicious events. The *TP* cases related to observed events remains the same. The same phenomenon occurs with the *FPs*, where an increase of only 2 is observed.

TABLE 6
Change in the Detection Results Observed on LADOHD When Evaluated With Both 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%

TABLE 7
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]

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) causes different outputs in the classification of an event. To this end, we work with the clean testing sequence of the Section 6.3.

Table 7 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. Fig. 5 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

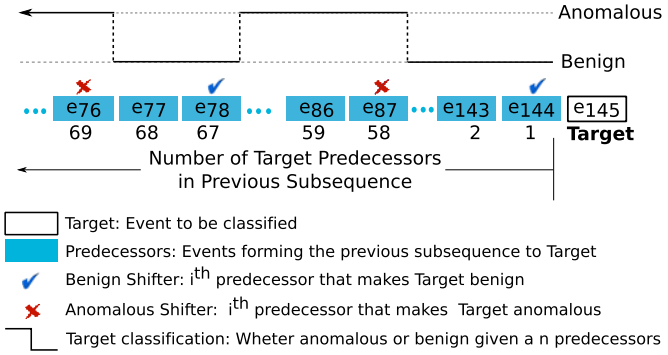


Fig. 5. Effect of long-term dependencies of LSTM models in the detection of anomalies. The example is based on the event e_{145} in Table 7.

the classification to be benign are called benign shifters. Those that cause the classification to be anomalous are referred to as anomalous shifters. Table 7 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 shows the potential of LSTMs 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 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.

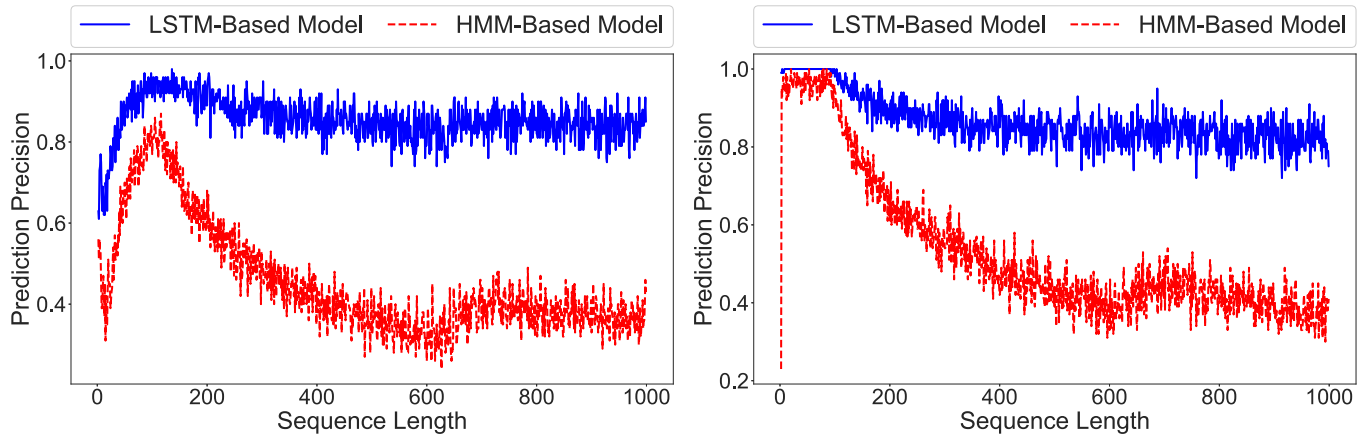
Findings. LADOHD can correctly classify events regardless the length of the previous subsequences. The history of events (hidden state of the model) is what actually affects the classification. The ability to look backward more than 600 steps (e.g., target e_{1032}) show the potential of this solution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

As the ability to discern between benign and anomalous events depends on the precision of the predictions made by the selected model, we want to study whether the complex structure of LSTM-based models provides an advantage when compared to other next-step-predicting methods able to handle variable length sequences such as HMM. We used the scikit-learn package to implement both a first-order full-connected HMM (i.e., number of hidden states equal to the number of all possible events) and a first-order left-to-right HMM (i.e., number of hidden states corresponds to the length of the training sequences) trained with the dataset described in Section 5. Despite that Yeung *et al.* [21] show that both configurations provide similar results, the full-connected configuration performed better in our evaluation. Hence, the presented results are based on this configuration. We acknowledge that a higher order HMM might produce more competitive results. However, we found ourselves limited in our implementation due to the complexity and computational cost imposed by the HMM algorithm.

We took 100 continuous subsequences of a specific length and measure the precision of the model predicting the last event in the sequences. The lengths were chosen to vary from 2 to 1000. To ensure a fair comparison, we measure the prediction precision of both models with incremental-length subsequences extracted from both the training and testing



(a) Precision Witn Training Sequences

(b) Precision With Testing Sequences

Fig. 6. Prediction accuracy of our LSTM and the HMM models with sequences of incremental lengths. Fig. 6a and Fig. 6b show the accuracy variation with subsequences extracted from the training and testing sequences respectively.

TABLE 8
Comparison of LADOHD to Lu's Method

Training Data Reduction	LADOHD (SeqLen = 10, k dynamic) TPR	FPR	F1-Score	k	Lu's Method (SeqLen = 10) TPR	FPR	F1-Score
0%	96.61%	0.25%	76.41%	12	89.15%	0.29%	70.04%
20%	96.39%	0.29%	74.24%	12	95.08%	0.32%	71.51%
40%	96.72%	0.24%	77.33%	15	77.38%	0.20%	69.92%
60%	96.45%	0.27%	75.89%	9	88.39%	0.35%	66.91%
80%	95.16%	0.32%	71.95%	8	96.13%	0.43%	66.52%

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 change 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. Fig. 6 shows that our LSTM-based model constantly outperform the prediction capacity of the HMM model. Although our model performs the best for subsequences of length < 105, it keeps predicting well with larger sequences, while the precision 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.

6.6 Comparison With a State-of-the-Art Detector

RQ6. How well does LADOHD perform compared to the LSTM-based method presented in [38]?

Table 8 shows the performance comparison of LADOHD to the detection method presented by Lu *et al.* [38]. This method is a LSTM-based solution that uses a fixed k parameter approach and operates with a pre-established number of predecessors N. Namely, given a sequence of length M, $M - N$ subsequences of length $N + 1$ are generated using a sliding windows mechanism to predict the last event in each subsequence. To compare Lu's method to LADOHD, we incrementally reduced the original training sequence (s_t) to train each model and evaluate their detection performance. In each round, we removed an additional 20% of the earliest events in the data. The goal of the experiment is to evaluate how well the models perform for different dataset sizes. We set the number of predecessors (N) for both models to 10 as proposed by Lu and coinciding with our results in experiment 6.5, where sequences of length $N + 1 < 105$ provided the best prediction precision. The value of the parameters k in Lu's method were found by an exhaustive search with the F1-Score metric, while it was dynamically set for LADOHD. Table 8 presents TPR, FPR, and F1-Score values of both methods, showing that LADOHD outperforms Lu's method in all the cases. A paired t-test between the F1-Score values of each method indicates that LADOHD's performance is significantly higher with a p-value of 0.0041.

Findings. LADOHD outperforms Lu's method for different training dataset sizes.

7 Conclusion and Future Work

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 from a controlled, real-world insider threat attack. 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 and its potential to be incorporated as a new analytics module to the detection stack of the EDR currently in production. Part of our future work is to evaluate the framework with different datasets and determine its runtime performance in a production environment. In addition, this work presents a comprehensive analysis of how LSTM-based models work and compare them to alternative solutions such as HMM-based models. We found that LSTM-based models rank the set of expected events in each timestep of variable-length sequences better than HMM-based models, which favor LSTMs in the detection of 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*. Berlin, Germany: Springer, 2008, pp. 69–90.
[2] A. Sanzgiri and D. Dasgupta, "Classification of insider threat detection techniques," in *Proc. 11th Annu. Cyber Inf. Secur. Res. Conf.*, 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 *Proc. IFIP Int. Conf. Commun. Multimedia Secur.*, 2014, pp. 63–72.
[5] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proc. 9th ACM Conf. Comput. Commun. Secur.*, 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 *Proc. Int. Workshop Recent Advances Intrusion Detection*, 2004, pp. 21–38.
[7] J. T. Giffin, S. Jha, and B. P. Miller, "Automated discovery of mimicry attacks," in *Proc. Int. Workshop Recent Advances Intrusion Detection*, 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 Inf. Secur. Privacy 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 *Proc. IEEE 28th Comput. Secur. Found. Symp.*, 2015, pp. 497–511.
[10] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Comput. Netw.*, 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," *Intell. Data Anal.*, 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 *Proc. IEEE Conf. Cybern. Intell. Syst.*, 2006, pp. 1–6.
[13] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. Hoboken, NJ, USA: Wiley, 2005, vol. 589.
[14] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, 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. Bellingham, WA, USA: SPIE, 2004, pp. 40–48.
[16] M. B. Salem and S. J. Stolfo, "Masquerade attack detection using a search-behavior modeling approach," Dept. Comput. Sci., Columbia Univ., New York City, NY, Tech. Rep. CUCS-027–09, 2009.
[17] C. Warrender, S. Forrest, and B. A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Proc. IEEE Symp. Secur. Privacy*, 1999, pp. 133–145.
[18] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 305–316.
[19] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proc. IEEE Symp. Secur. Privacy*, 1996, pp. 120–128.
[20] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, 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 Recognit.*, 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 *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 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 *Proc. Advances Neural Inf. Process. Syst.*, 1999, pp. 889–895.
[24] P. Smyth, "Clustering sequences with hidden Markov models," in *Proc. Advances Neural Inf. Process. Syst.*, 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 *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discov. 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 Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.
[27] G. Salton, R. Ross, and J. Kelleher, "Attentive language models," in *Proc. 18th Int. Joint Conf. Natural Lang. Process.*, 2017, pp. 441–450.
[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[29] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1724–1734.
[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 *Proc. 13th Annu. Conf. Int. Speech Commun. Assoc.*, 2012, pp. 194–197.
[32] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Proc. Advances Neural Inf. Process. Syst.*, 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," 2016, *arXiv:1611.01726*.
[34] C. Feng, T. Li, and D. Chana, "Multi-level anomaly detection in industrial control systems via package signatures and LSTM networks," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 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 *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.
[36] Y. Shen, E. Mariconti, P.-A. Vervier, and G. Stringhini, "Tiresias: Predicting security events through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 592–605.
[37] F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang, "Insider threat detection with deep neural network," in *Proc. Int. Conf. Comput. Sci.*, 2018, pp. 43–54.
[38] J. Lu and R. K. Wong, "Insider threat detection with long short-term memory," in *Proc. Australas. Comput. Sci. Week Multiconf.*, 2019, pp. 1–10.
[39] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.
[40] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015, *arXiv:1506.00019*.
[41] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Reading, MA, USA: Addison-Wesley, 2012.
[42] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," in *Proc. Int. Conf. Learn. Representations*, 2018.
[43] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Representations*, 2017.
[44] L. N. Smith, "Cyclical learning rates for training neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2017, pp. 464–472.
[45] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," The MITRE Corporation, McLean, VA, USA, Tech. Rep., 19-01075-28, 2020.
[46] C. Wueest, S. Doherty, and H. Anand, "The increased use of powershell in attacks," *Symantec Corporation*, Mountain View, CA, USA, Version 1.0, 2016.

Miguel Villarreal-Vasquez received the PhD degree in computer science from Purdue University, West Lafayette, IN, in 2020. He is currently a senior AI/ML engineer at JPMorgan Chase, Seattle, WA, USA.

Gaspar Modelo-Howard (Senior Member, IEEE) received the PhD degree in computer engineering from Purdue University, West Lafayette, IN, in 2013. He is currently a senior principal data scientist with the public Cloud security team at Palo Alto Networks, Santa Clara, CA, USA.

Simant Dube received the PhD degree in computer science from University of South Carolina, Columbia, SC, in 1992. He is currently a technical director in the Center for Advanced Machine Learning (CAML) at Symantec, Mountain View, CA, USA.

Bharat Bhargava (Fellow, IEEE) received the PhD degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 1974. He is currently a professor of computer science at Purdue University.