Improving Host-based Intrusion Detection Using Thread Information

Martin Grimmer[⋆], Tim Kaelble[⋆⋆], and Erhard Rahm[⋆ ⋆ ⋆]

Leipzig University, ScaDS.AI, Humboldtstraße 25, 04105 Leipzig, Germany
https://scads.ai/

Abstract. Host-based anomaly detection for identifying attacks typically analyzes sequences or frequencies of system calls. However, most of the known approaches ignore the fact that software in modern IT systems is multithreaded so that different system calls may belong to different threads and users. In this work, we show that anomaly detection algorithms can be improved by considering thread information. For this purpose, we extend seven algorithms and comparatively evaluate their effectiveness with and without the use of thread information. The evaluation is based on the LID-DS dataset providing suitable thread information.

1 Introduction

The frequency of cyber attacks and the associated damage are steadily increasing. This results in the interruption of various lines of business and services causing enormous financial losses (about 5.2 trillion USD) as shown in an Accenture study in 2019 [2]. In the event of loss of personal data, further financial penalties may be imposed on companies within the European Union due to the General Data Protection Regulation (GDPR) if adequate protection measures have not been taken, as required in Section 2 (Security of personal data), Article 32 (Security of processing) of the GDPR [8]. Another important point is the potential loss of customers' trust by losing their personal data. Restoring lost trust is a very long and difficult process. This all gets worse if you consider that in the past, on average, companies did not notice security incidents until weeks later as stated in the 2018 U.S. State of Cybercrime report [14]. For small and

⋆ grimmer@informatik.uni-leipzig.de
⋆⋆ tim_k@posteo.de
⋆ ⋆ ⋆ rahm@informatik.uni-leipzig.de
Acknowledgment: This work was supported by the German Federal Ministry of Education and Research within the project Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS18026B) and by the German Federal Ministry for Economic Affairs and Energy (BMWi - ZIM, 16KN061130) based on a resolution of the German Parliament within the project "Anomaly-based attack detection on data- and control flow-based sensors". Computations for this work were done with resources of Leipzig University Computing Centre.

medium-sized companies, on average 56 days elapse between the time they notice a safety incident and the time when the incident is suspected to have occurred. For large enterprises, this time span is even 151 days on average.

A first step towards better IT security is to detect security incidents promptly. Only then can appropriate countermeasures be taken and potential security gaps be closed quickly. A possible technical solution for the fast identification of attacks is the use of intrusion detection systems. They can also help to address the requirements stated in article 32 (b) GDPR: *"Taking into account the state of the art [...], the controller and the processor shall implement appropriate technical [...] measures to ensure a level of security appropriate to the risk, [...] the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services [...]."* [8]

Even if intrusion detection systems could only detect and avert 5% of all IT security incidents in time, this would correspond to an avoided loss of about 260 billion USD worldwide. It is therefore also worthwhile to examine approaches to enhance Intrusion Detection Systems (IDS).

1.1 Our Contribution

The aim of this work is to improve existing approaches for host-based intrusion detection systems (HIDS), in particular, anomaly-based approaches that evaluate system calls. These approaches mostly analyze sequences or frequencies of system calls. However, they do not consider the thread in which a system call has been executed, thereby ignoring the fact that software in modern IT systems is mostly multithreaded. Multiple runs of the same software, with identical input data, may thus result in different sequences of system calls due to the execution scheduling of the operating system. Since much current work on HIDS is focused on learning and analyzing subsequences of system calls, this can lead to incorrect results. This paper aims at answering the question whether considering the thread allocation of a system call can achieve better results in anomaly detection. For this purpose, we modify and evaluate seven known anomaly detection algorithms accordingly. We will show that the use of thread information increases the detection rate and reduces the number of false alarms in most cases. Our analysis also focuses on streams of system calls for a fast identification of anomalies while previous work mostly analyzes entire files in a batch-like manner.

1.2 Paper Outline

The remainder of this paper is structured as follows: First, we introduce a categorization of IDS to put the approaches of this paper into context. Then we present previous work for host-based anomaly detection. Following this, we discuss the available datasets and how they can be used to provide valid inputs for the algorithms. Afterwards we briefly introduce the considered algorithms and discuss how to determine a threshold for deciding whether an observation is normal or abnormal. We then describe the experimental setup, discuss the results of the experiments, and draw conclusions.

2 Host-based Intrusion Detection

2.1 Categories of Intrusion Detection Systems

According to Milenkoski et. al. [25] and Debar et al. [6] Intrusion Detection Systems can be categorized along the dimensions: the monitored platform, the method to detect attacks and the deployment architecture.

The Monitored Platform – If network data is evaluated, one speaks of network-based intrusion detection systems (NIDS). These systems analyse the incoming, outgoing and internal traffic of a network and can thus evaluate the externally "visible" behaviour of the systems within the monitored network. If, instead of network data, activities of the systems to be monitored are collected, this is referred to as host-based intrusion detection systems (HIDS). These systems have a vast amount of different data sources at their disposal to detect intrusions. Thus, characteristics such as CPU load, memory consumption or application logs can be examined. Another comprehensive and detailed source of the behavior are system calls executed on the host.

The Attack Detection Method – An intrusion detection system that checks monitored activity against a database of known attacks or predefined rules is called misuse- or signature-based. Such systems are not able to identify so-called zero-day attacks, i.e. attacks that were previously unknown. Unlike misuse-based systems, anomaly-based systems, also called behavior-based systems, detect deviations from a known normal behavior. They do not require attack signatures in advance allowing them to detect previously unknown attacks. The literature emphasizes that anomaly-based IDS often misinterprets normal behavior as anomalous, which is the greatest of their drawbacks [25].

The Deployment Architecture – Systems consisting of several subsystems that are used at different locations and communicate with each other to detect intrusions are called compound IDS. They can also detect coordinated attacks that have multiple targets. Otherwise they are called non-compound, i.e., IDS at a single location

Categories Considered in this Paper – In this paper, we focus on HIDS that can make use of rich information about the internal system behavior. We further focus on anomaly-based IDS as they can also detect previously unknown attacks. As mentioned before, they often have to contend with high error rates. Therefore we want to achieve low error rates and investigate whether this can be achieved by considering metadata such as the thread information of system calls as already suggested by Pendleton and Shouhuai in [29]. As a first step we want to investigate this for the simpler case of non-compound IDS and leave the more complex case of compound IDS for future work. Therefore we will deal with algorithms of non-compound anomaly-based host IDS.

2.2 Related Work

In 1996, an initial approach to intrusion detection called TIDE (time-delay embedding) was proposed based on a database of simple valid patterns (lookahead pairs) of system calls [9]. Using these patterns, the proportion of known patterns is determined to decide about whether a given trace is anomalous or not. STIDE (sequence time-delay embedding) expands this approach by considering contiguous sequences of system calls of fixed length instead of lookahead pairs [13]. Further work dealt with different evaluation possibilities of the underlying concept [33] [11]. In 2001, Eskin proposed a method with dynamic window sizes using entropy modeling and context dependency to determine the optimal window size. [7] Another approach to eliminate the need of a fixed window size was introduced by Marceau using a finite state machine whose states represent predictive sequences of different lengths. [22] Jewell and Beaver investigated whether making the normal profile dependent on the user (uid) makes a difference [15]. They also considered dynamic sizes of system call sequences by dividing the input into sequences in which each system call occurs exactly once. In doing so, they defined the system calls for these types of sequences as (syscall, errno, args). [15] Since sequence-based approaches usually incur high computational and memory overheads, researchers also looked at the use of frequencies of individual system calls [16] [1]. While these features are more lightweight in their computation, they are also less detailed so making it difficult to achieve similarly good detection and false alarm rates as sequential approaches. Other possibilities for feature extraction to improve anomaly detection include the use of parameter values of individual system calls. Kruegel et al. [19] created a separate model for each system call (e.g., write or open), which contains specific analyses for the parameters used. The model considers strings, characters, structure or tokens used. This approach was later extended with sequence analysis in 2006 and 2008 [28] [21]. Many of the mentioned papers try to determine whether an attack occurs within a given trace which corresponds to a file of the used dataset. These papers determine their detection and false alarm rates accordingly at file level [9] [13] [11] [7] [16] [19] [28]. Some other works show that it can be done more detailed and determine the exact number of alarms and false alarms [33] [22] [15] [1] [21].

3 Datasets

Over the last two decades several datasets for evaluating HIDS using system calls have been published. The best known of these datasets include: The *KDD dataset*, to be more precise the BSM (Basic Security Module) of the DARPA Intrusion Detection Evaluation Data Set, from 1998 to 2000 [20], the Intrusion Detection Data Set from the University of New Mexico known as *UNM dataset* from 1999 [4] [34], the data sets of the Australian Defence Force Academy, the *ADFA-LD* from 2013 [5] [3] and the *NGIDS-DS* from 2017 [12].

All of the above mentioned datasets have critical drawbacks as discussed in [29] and [10]. The KDD and UNM datasets are outdated based on software older than 20 years that is hardly in use anymore. Furthermore, the hardware and the corresponding operating systems with their system calls have changed during this time. ADFA-LD is also several years old and lacks thread information. Since this paper deals with the question whether thread allocation of a system call can improve anomaly detection, this kind of information is needed in a suitable evaluation dataset. Only the NGIDS-DS and the newer LID-DS datasets [10] [30] meet this requirement and are up to date. Therefore, these two datasets are considered further in this paper.

3.1 NGID-DS

The Next Generation Intrusion Detection System - Data Set (NGIDS-DS) consists of labeled network and host (system call) logs aiming to realistically reflect critical cyber infrastructures of enterprises in both normal and abnormal scenarios [12].

The Sequence Problem – In the host files of the NGIDS-DS each entry has a timestamp accurate to the second and an event id which indicates the order of the executed system call on the host system. Unfortunately we found that it is not possible to reliably determine a deterministic order of the system calls using this information. This is because there are in general several calls with the same timestamp and furthermore the order of event-ids sometimes contradicts the order of timestamps. We found that there are contracting entries in the files of NGIDS-DS, an example is shown in Table 1. Here the first entry has a smaller timestamp than the second one, while the event-ids are ordered the other way around. Therefore, unfortunately, it is not possible for us to reconstruct the correct sequence of system calls. Since all algorithms considered in this paper analyze the sequence of system calls, we cannot use the NGIDS-DS data.

Table 1. Extract from NGIDS-DS host file 1.csv with conflicting timestamp and event-id orders.

date	time	pid	process name	syscall	event id
11/03/2016	2:45:01	2114	/usr/bin/compiz	168	45357
11/03/2016	2:45:06	1804	/bin/dbus-daemon	256	45352

3.2 LID-DS

To address the observed shortcomings of the older datasets, Grimmer et al. created the LID-DS. It contains system calls, timestamps, thread ids, process names, arguments, return values and excerpts of data buffers from traces of normal and attack behaviour of several recent, multi-process, multi-threaded scenarios. Many of the included features are not available from previous datasets [10]. For this paper, the system calls, their unambiguous sequence and their thread assignment are of particular importance.

Structure of the LID-DS – The LID-DS consists of 10 different scenarios each representing a real vulnerability. They are named after the official Common Vulnerabilities and Exposures (CVE) [27] number, the corresponding Common

Table 2. LID-DS scenarios, their abbreviation and their number of system calls.

scenario	abbr.	#syscalls	scenario	abbr.	#syscalls
Bruteforce-CWE-307	BF	5 696 050	CVE-2012-2122	2012	5 721 512
CVE-2014-0160	2014	4 009 668	CVE-2017-7529	2017	1 796 862
CVE-2018-3760	2018	19 160 009	CVE-2019-5418	2019	17 955 534
EPS-CWE-434	EPS	126 458 405	PHP-CWE-434	PHP	22 268 842
SQL-Injection-CWE-89	SQL	23 616 570	ZipSlip	ZIP	252 934 566

Weakness Enumeration (CWE) number [26] or a specific name (e.g. ZipSlip). Each of these scenarios contains about 1137 files consisting of about 1021 normal and 116 attack sequences. The attack sequences consists of both normal behavior and attack behavior. What this means for the evaluation of the experiments will be explained in Section 6. The recordings are between 30s and 60s long (45s on average). The first 200 normal sequences are used as *training data*, the next 50 are used as *validation data* and the remaining normal and attack sequences are used as *test data*. This leaves about 771 normal sequences ($771 \cdot 45s$) and about half the runtime of the attack sequences ($116/2 \cdot 45s$) that contain normal behavior. This corresponds to roughly 10 hours of runtime of normal behavior per scenario.

In the remainder of this paper, we abbreviate the individual scenarios of the LID-DS. The original names, our abbreviation and the number of system calls in the scenarios can be found in Table 2.

4 Feature Engineering

In the following, we present different concepts and preprocessing methods needed for the anomaly detection approaches.

n-gram – A n-gram is a contiguous sequence of n items, e.g., system calls, from a given input. If for example the sequence (a, b, c, d) is given, then following n-grams can be derived for n=2: (a,b), (b,c) and (c,d).

n-grams using thread information – As mentioned before many IT systems nowadays are multi-threaded. As a result, multiple runs of the same software with identical input data can lead to different sequences of system calls across all threads due to the execution scheduling of the underlying operating system. Ignoring this fact as done in previous approaches can thus lead to incorrect results for anomaly detection. This is illustrated by the example in Table

Table 3. Two traces resulting from different runs of the same software and input in a multi-threaded environment.

thread t_1:		b			d			b
thread t_2:			c			b		
thread t_3:	a			a			c	
flat sequence s:	a	b	c	a	d	b	c	b

thread t'_1:			b		d			b
thread t'_2:		c				b		
thread t'_3:	a			a			c	
flat sequence s':	a	c	b	a	d	b	c	b

3 where threads 1 and 1', 2 and 2', and 3 and 3', respectively each perform the same sequences of system calls. However, due to different scheduling, the resulting so-called *flat sequences* s and s' (the system call sequences without thread information) differ from each other.

Ignoring this information can lead to learning incorrect data. As shown in Table 4 the resulting n-grams (here with $n = 2$) differ depending on the source sequence. The left side shows the n-gram frequencies for the flat sequence s, the middle shows the n-gram frequencies for flat sequence s' and the right side shows the frequencies in case they were determined from the underlying threads t_1, t_2 and t_3 (which is the same for t'_1, t'_2 and t'_3). For example, n-gram or subsequence (b c) occurs twice in flat sequence s but only once in s' and never in the thread-specific sequences.

Table 4. Frequencies of n-grams (n=2) from sequences s, s' and t_1, t_2, t_3.

	from s				from s'				from t_1, t_2, t_3			
	a	b	c	d	a	b	c	d	a	b	c	d
a	0	1	0	1	0	0	1	1	1	0	1	0
b	0	0	2	0	1	0	1	0	0	0	0	1
c	1	1	0	0	0	2	0	0	0	1	0	0
d	0	1	0	0	0	1	0	0	0	1	0	0

One Hot Encoding of Categorical Data – Machine learning (ML) algorithms usually require numerical (quantitative) inputs. However, system calls and n-grams of system calls are nominal qualitative (categorical) data but not quantitative information. One way to represent this type of data for an ML algorithm is one hot encoding (OHE). To represent a system call c_i from the set of m possible system calls $C = \{c_1, c_2, ..., c_m\}$, a one-hot vector v_i of length m is used. For system call c_i such a vector v_i is a bit vector with value 1 at position i and value 0 in all other positions. n-grams of system calls can be encoded by concatenating the one hot encoded system calls.

System Call Embeddings – The use of long OHE representations can lead to a large number of weights to be trained in neural networks. Therefore it is advisable to use shorter vectors. In the field of natural language processing, so called word embeddings have proven their value, e.g., based on the Word2Vec method proposed by Mikolov et al. [24]. Word2vec uses a neural network to learn word associations from a large corpus of text. A word is thereby mapped to a vector of a certain length, the word embedding. The cosine similarity between two such vectors indicates the degree of semantic similarity between the associated words. We have applied this principle to system calls. For each scenario of the LID-DS, we generated a system call corpus of the training data. We then used this to compute the corresponding word embeddings using the original implementation of Mikolov et al. [23]. We have calculated two variants of word embeddings. The first, "thread unaware" variant uses the system calls as they occur in the training data one after the other. The second, "thread aware" variant considers the system calls of the training data per thread, so that the system calls of different threads do not mix. With the system call embeddings computed in this way, the composite n-grams can be generated by simple concatenation, as before. We later call this embedding W x, where x corresponds to the length of the vector.

Duplicates in the Training Data – In this work, we decided to use the training data not as presented, but duplicate free. For all algorithms we discuss in Section 5, except SCG (since it should explicitly learn the distribution of node transitions) the training data is modified in a way, such that duplicate entries of the same n-grams are removed from the input to the training algorithm. This is done to reduce the size of the training matrices (and their used memory) and to speed up training time. By removing duplicates in the training data, the algorithms are trained in such a way that frequent and rare "patterns" in the training data are learned "equally" well.

5 Algorithms

In this paper we consider only algorithms or variants of algorithms that can process streams of system calls to enable a fast identification of anomalies at runtime. Hence we do not investigate a more batch-like retrospective analysis of entire collections of system calls. To process a stream of system calls, each algorithm always receives exactly one system call after another as input. It then builds n-grams as described above within a given **streaming window** of a predefined length. It then calculates a so-called **anomaly score** indicating how abnormal the determined n-gram is in relation to previously seen system call n-grams.

Each of the algorithms presented here basically consists of two phases: the training phase and the detection phase. In the following descriptions, we will discuss these two phases separately. In addition, for each of the algorithms, we specify how the final anomaly value is calculated for a streaming window.

The Bag of System Calls (**BOSC**) algorithm was introduced in [16] and later applied in [1]. It does not consider the order of system calls in a n-gram but only the frequency of occurrence.
Training: N-grams of the training data are built. Then the occurrences of every system call within each n-gram is counted. A bag of system calls in this sense is the assignment from all system calls to their frequency within a given n-gram. All these bags are then saved in a database for later use as normal bags.
Detection: In detection mode every n-gram is converted to a bag as described. Then for each bag it is checked if it is included in the normal database from the training phase. If the bag is included, we call it a match otherwise a mismatch.
Anomaly: An anomaly is detected if the anomaly score (ratio of mismatches to number of n-grams in the streaming-window) is greater than a predefined threshold.

In 1998 Hofmeyer et al. designed the Sequence Time-Delay Embedding (**STIDE**) [13] algorithm.

Training: The STIDE algorithm is similar to the BOSC algorithm, but rather looking only at the frequencies and ignoring the sequence of system calls the actual n-grams are stored and used as the normal database.
Detection: In the detection phase the STIDE algorithm checks whether the current n-gram is included in the normal database. Similar to the BOSC algorithm if the n-gram is included, we call it a match otherwise a mismatch.
Anomaly: As before an anomaly is detected if the anomaly score (ratio of mismatches to the number of n-grams in the streaming-window) is greater than a predefined threshold.

The method of System Call Graphs (**SCG**) is also called n-gram probability graph [11]. Here system calls are placed in a directed weighted graph, where the nodes describe n-grams of system calls and the edges represent the probabilities of transitions from one n-gram to another.
Training: In the training phase the n-gram probability graph is built as described above.
Detection: In the detection phase the transition probability from one n-gram to its successor n-gram is determined using the n-gram probability graph. If this transition is not present in the graph its probability is set to 0.
Anomaly: The resulting anomaly score is then given by the mean of all transition probabilities within the streaming window. If this anomaly score is greater than a predefined threshold an anomaly is detected.

The One Class Support Vector Machine (**SVM**), introduced in [31] and later applied in [35] is a variant of the Support Vector Machine. Simply put it tries to learn a function which classifies if a sample could have been drawn from the input distribution.
Training: The One Class SVM is trained for all n-grams from the training data.
Detection: The One Class SVM algorithm decides whether a given n-gram is part of the normal class (match) or not (mismatch).
Anomaly: As before an anomaly is detected if the anomaly score (ratio of mismatches to the number of n-grams in the streaming-window) is greater than a predefined threshold.

Different kinds of artificial neural networks are used to predict the next element of a given sequence. This for example was done in [36] on sequences of numbers and in [17] on sentences. We want to transfer the principle applied there to our application. Here it is used to predict the probability of each possible next system call after a given n-gram of system calls. We utilize two variants of artificial neural networks: The Multi Layer Perceptron (**MLP**) and Convolutional Neural Networks (**CNN**) which are known for their ability to identify patterns in data. As the two approaches just differ in the neural network architecture they are jointly summarized in the following.
Training: N-grams of system calls are being used as the input of the MLP/CNN. The expected output is the successor system call. The actual successor system call can be used for supervised learning.
Detection: With the given n-gram, the probability of every possible system call appearing in the next time step is predicted.

Anomaly: The anomaly score (one minus the mean probabilities of all n-grams in the streaming window) is then again compared to a predefined threshold. If it is greater than this threshold we count it as anomaly.

Autoencoders (**AE**) can learn to compress data through finding correlated features in the data. [18] With these features they are capable of reconstructing the compressed data with less noise. If the underlying features which it learned are less represented in the test data the reconstruction loss is greater. So with reconstructing an anomaly the loss is expected to be higher than with data from the training data.
Training: The Autoencoder tries to learn the features of all n-grams of the training data. By reconstructing this data through the Autoencoder, a loss can be determined which should be minimized.
Detection: The mentioned reconstruction loss between input and output of the Autoencoder is then used as the anomaly score of the specific n-gram.
Anomaly: If the average anomaly score of all n-grams in a streaming window is higher than a predefined threshold, an anomaly is detected.

6 Evaluation approach

Before we come to the results of our investigation in the next section, we want to first outline how we determine the sucess of anomaly detection with streams of system calls and how we set the anomaly thresholds.

Anomaly Evaluation – As described before each algorithm produces an anomaly score for every new system call in a stream. How we interpret this score depends on the threshold value as well as the attack start time. The threshold value is used to distinguish between normal classifications (the anomaly score is less or equal the threshold) and abnormal classifications (the anomaly score is greater than the threshold) as described before. The expected attack start time t_{attack} is recorded in the LID-DS for each file containing an attack. Unfortunately t_{attack} is somewhat imprecise so that we soften this value a bit and use $t'_{attack} = t_{attack} - 2s$ instead. As visualised in Figure 1, using the anomaly threshold and attack start time we obtain four areas (quadrants) that can be used for evaluation anomaly detection. In particular, one can classify a given anomaly score as a false alarm, FA (or FP, false positive), i.e., incorrectly classified as attack, if the score is in quadrant 2. An anomaly or false alarm may span more than one system call or n-gram. Therefore, an alarm should not be triggered for every single abnormal system call or every single abnormal n-gram. In reality, this would result in a too large number of alarms that have to be manually reviewed. Instead, alarms that directly follow each other should only counted as a single alarm. A score in quadrant 3 counts as normal behavior (TN, true negative). Unfortunately, its hard to make such statements for values with a timestamp greater then t'_{attack} since we don't know whether the attack has already started, it is already over and if in this case normal behavior has appeared again already. The only thing we know for this case is that after t'_{attack} an attack is expected. Therefore, to correctly classify an attack (TP, true positive) at least one anomaly score in quadrant 1 is needed for an attack file. Due to the given expected start times of all attacks in the LID-DS, we can determine false alarms in the files containing attacks up to t'_{attack}. This allows much more accurate statements than simply classifying an entire file as benign or malicious, as done for other datasets before. Additionally we calculate the detection rate $DR = TP/A$ with A, the number of attacks in the test data of a scenario, to improve the comparability of the individual scenarios, since they do not all contain the same number of attack files. To compare the results in this work we use DR and FA.

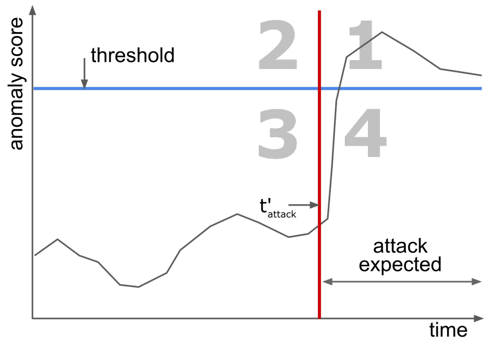

Fig. 1. Exemplary plot of anomaly scores (y axis) against time (x axis). It shows the threshold (horizontal blue line), attack start time (vertical red line) and the resulting four quadrants 1-4.

Determining the Threshold Value – The choice of the anomaly threshold obviously is a critical parameter that impacts the overall quality of anomaly detection. We aimed at automatically finding suitable values. Given that anomalies are unknown beforehand we only use the data about normal behavior for this purpose. In particular, we use the maximum anomaly score from the corresponding validation part of the normal behavior as the threshold for the normal class. By using the validation data, we prevent overfitting compared to using the actual training data to determine the threshold.

Experiments – To investigate how the described thread information affects the different algorithms presented, we ran a grid search over a large set of possible configurations and all scenarios of the LID-DS. In total, we have trained and evaluated more than 30,000 different configurations of the presented algorithms.

7 Findings

In this section, we analyze how well the presented algorithms perform on the LID-DS without and with using thread information. We also analyse the impact for different scenarios.

7.1 Results Per Algorithm Over All Scenarios

We first evaluate the impact of using thread information (*TI*) on the average performance of the individual algorithms over all scenarios. For this purpose, we report for every algorithm results for three configurations, (1) the configuration achieving the best detection rate (DR) without thread information, (2) the results for the same configuration of (1) but with the use of thread information (i.e., after switching the "thread information flag"), and (3) the best configurations with thread information. The obtained results over all scenarios are summarized in Table 5 together with the used encodings for n-grams and the considered n-gram lengths n and window sizes l.

We observe that the best configuration using thread information (conf. 3) achieves a better detection rate for six of the seven algorithms and about the same detection rate for the autoencoder approach (AE). For BOSC and STIDE, even just switching the TI flag for the best non-TI configuration achieves a noticeable DR improvement despite the fact that for the use of threads the window size should be far smaller (1000 vs. 10000). The FA results for the number of false alarms are mixed. Here using thread information results in improvements for four of the seven algorithms (SCG, MLP, CNN and AE). Hence for these algorithms the use of thread information improves (or maintains) both detection rate and the number of false alarms. The overall best detection rates are achieved for the BOSC and the STIDE approaches (98,6%) and the lowest number of false alarms for AE and MLP.

We now want to illustrate the relative performance of the best algorithms for the different scenarios in more detail. Since it is difficult to rank configurations for multiple target criteria (DR, FA) one can use the concept of Pareto optimality, a condition in which no criterion can get better without making at least one other criterion worse. According to this concept, the configurations BOSC 1, STIDE 3, SCG 3, MLP 3 and AE 3 from Table 5 are Pareto optimal. Therefore in Tables 6 and 7 we show the results of these configurations and their corresponding configuration with altered TI flag for each scenario.

We observe that the different algorithms behave quite differently for the various scenarios. The STIDE approach excels in achieving high detection rates for all scenarios with use of *TI*. In contrast to the other algorithms and with the use of *TI*, STIDE and SCG are especially able to solve scenario 2014 with some and ZIP with only one respectively zero FA. On the downside, for STIDE and in particular BOSC using *TI* leads to relatively high FA values. From one scenario solved to nine, with almost no change in FA, the SCG algorithm also benefits greatly from *TI*. The two other algorithms achieve high detection rates only for six to eight of the ten scenarios. The MLP algorithm benefits strongly from using *TI*, now able to solve eight instead of four scenarios and fewer false alarms compared to not using thread information. Perfect results (DR 1, FA 0) are achieved for the 2019 and EPS scenarios. Finally, the AE approach achieves perfect detection rates in three scenarios. The use of *TI* does not affect DR, but it does reduce the number of false alarms, especially in the PHP, 2018 and SQL scenarios.

Table 5. Algorithm wise best average configuration over all scenarios. The columns are: algorithm, configuration, encoding, thread information, n-gram length, streaming window length, mean DR and mean FA.

alg.	con.	enc.	*TI*	n	l	$\overline{DR}$	$\overline{FA}$
BOSC	1	-	-	11	10000	0.885	17.4
BOSC	2	-	+	11	10000	0.949	519.5
BOSC	3	-	+	7	1000	0.986	110.4
STIDE	1	-	-	3	10000	0.879	12.5
STIDE	2	-	+	3	10000	0.981	118.6
STIDE	3	-	+	5	1000	0.986	61.5
SCG	1	-	-	9	10000	0.711	43.9
SCG	2	-	+	9	10000	0.686	166.4
SCG	3	-	+	5	10	0.895	29.1
SVM	1	OHE	-	5	100	0.562	13.3
SVM	2	OHE	+	5	100	0.292	13.3
SVM	3	OHE	+	9	100	0.595	42.7
MLP	1	OHE	-	11	100	0.614	18.1
MLP	2	OHE	+	11	100	0.605	34.7
MLP	3	OHE	+	7	1	0.789	4.2
CNN	1	OHE	-	7	100	0.686	35.4
CNN	2	OHE	+	7	100	0.694	28.2
CNN	3	OHE	+	22	10	0.702	20.0
AE	1	W 5	-	5	1	0.629	16.4
AE	2	W 5	+	5	1	0.622	2.3
AE	3	W 5	+	7	1	0.622	2.2

Table 6. Results for configuration 1 (without TI) and 2 (with TI) of BOSC and configuration 3 (with TI) and its variant without TI of STIDE.

	BOSC				STIDE			
	no TI (1)		with TI (2)		no TI		with TI (3)	
scenario	DR	FA	DR	FA	DR	FA	DR	FA
BF	0.94	32	0.94	29	0.94	66	0.94	24
2012	0.95	11	0.99	201	0.03	4	0.99	26
2014	0.00	1	0.59	210	0.00	3	0.95	44
2017	0.97	9	0.99	5	0.97	9	0.99	5
2018	0.99	19	0.99	5	0.99	25	0.99	12
2019	1.00	4	0.99	22	1.00	8	1.00	18
EPS	1.00	20	1.00	604	1.00	4	1.00	12
PHP	1.00	2	1.00	119	0.95	1	1.00	93
SQL	1.00	40	1.00	3183	0.15	18	1.00	380
ZIP	1.00	36	1.00	817	0.34	1	1.00	1
mean	0.89	17.4	0.95	519.5	0.64	13.9	0.99	61.5

Table 7. Results for configuration 3 (with TI) and its variant without TI of SCG, MLP and AE.

	SCG				MLP				AE			
	no TI		with TI (3)		no TI		with TI (3)		no TI		with TI (3)	
scenario	DR	FA	DR	FA	DR	FA	DR	FA	DR	FA	DR	FA
BF	0.04	2	0.94	1	0.00	0	0.94	1	0.00	1	0.00	0
2012	0.01	25	0.10	67	0.01	16	0.98	10	0.03	19	0.00	0
2014	0.01	33	0.95	25	0.00	0	0.01	7	0.00	0	0.00	0
2017	0.97	17	0.97	9	0.97	11	0.97	14	0.97	8	0.97	8
2018	0.00	4	0.99	35	0.00	0	0.99	2	0.99	79	0.99	11
2019	0.00	9	1.00	11	1.00	36	1.00	0	1.00	0	1.00	3
EPS	0.27	5	1.00	40	0.00	0	1.00	0	1.00	0	1.00	0
PHP	0.02	0	1.00	19	1.00	31	1.00	6	1.00	169	1.00	0
SQL	0.09	113	1.00	84	1.00	31	1.00	2	1.00	25	1.00	0
ZIP	0.03	12	1.00	0	0.00	0	0.00	0	0.27	0	0.27	0
mean	0.14	22.0	0.90	29.1	0.40	12.5	0.79	4.2	0.63	30.1	0.62	2.2

7.2 Scenario Wise Best Results

We now analyze the influence of using thread information for each of the ten scenarios separately. Table 8 shows the best result regarding DR per scenario without and with the use of *TI* together with the applied configuration. For cases of several configurations with the same "best" result for a scenario, only one is shown. The table shows that in nine of the ten scenarios the use of *TI* results in a better or about equally good DR and FA values (only for scenario "2018" the FA value is increased). In general, the use of thread information enables very good solutions for all scenarios with perfect or almost perfect detection rates (0.94 to 1.0) and 0 false alarms for 6 of the ten scenarios. Compared to the configurations not using thread information, DR is especially increased for scenario 2014 (from 0.06 to 0.95) and the FA value is improved or kept at 0 for nine scenarios. Table 8 also reveals that the AE algorithm is able to provide a perfect result with 100% DR and no FA for 4 scenarios, for both without and with using TI. For using *TI*, the ZipSlip scenario can also be perfectly solved by the BOSC algorithm (as well as SCG, not shown in the table). Overall the best results are mostly achieved by the AE, CNN, MLP and STIDE algorithms while the SVM approach is not among the top-performing approaches.

7.3 Overall Best Practical Configurations

The presented results show that in many cases it is not possible to achieve both very high detection rates and a very low number of false alarms with the same configuration over all scenarios. However configurations optimizing only one of the two goals are not sufficient and can be ineffective in practice. This is especially the case when IDS alarms need to be handled by a security expert to prevent possible damage to the monitored system. This asks for reducing the number of false alarms as much as possible while aiming for a detection rate as high as possible. For eaxample a configuration with $DR = 0.80$ and

Table 8. Scenario-wise best configurations over all algorithms without and with *TI*. The columns are: scenario, algorithm, encoding, thread information, n-gram length, streaming window length, DR and FA.

scenario	alg.	enc.	*TI*	n	l	DR	FA
BF	CNN	OHE	-	9	100	0.95	97
BF	MLP	OHE	+	5	1	0.94	0
2012	AE	OHE	-	5	100	0.97	33
2012	STIDE	-	+	5	1000	0.99	26
2014	MLP	OHE	-	22	10000	0.06	22
2014	STIDE	-	+	9	10	0.95	18
2017	CNN	OHE	-	5	100	0.99	13
2017	CNN	OHE	+	9	100	0.99	4
2018	CNN	OHE	-	7	100	1.00	5
2018	MLP	OHE	+	7	100	1.00	28
2019	AE	W 2	-	3	1	1.00	0
2019	AE	W 2	+	3	1	1.00	0
EPS	AE	W 2	-	3	1	1.00	0
EPS	AE	W 2	+	7	1	1.00	0
PHP	AE	W 2	-	3	1	1.00	0
PHP	AE	W 2	+	3	1	1.00	0
SQL	AE	W 2	-	3	1	1.00	0
SQL	AE	W 2	+	3	1	1.00	0
ZIP	STIDE	-	-	5	10000	1.00	4
ZIP	BOSC	-	+	3	10	1.00	0

$FA = 10$ might be more effective than one with $DR = 0.90$ and $FA = 100$ or even $DR = 0.99$ and $FA = 1000$ since manually dealing with 100 or even 1000 false alarms can make the system impractical to use.

At this point the concept of Pareto optimality shall be applied again. One possible approach to limit the number of possible Pareto-optimal results is to define certain levels of the accepted number of false alarms and ranking the configurations according to their DR results within each level. For example, we might be interested in the following five FA levels: Level 1: $\overline{FA} < 40$, level 2: $\overline{FA} < 20$, level 3: $\overline{FA} < 10$, 4: $\overline{FA} < 5$ and 5: $\overline{FA} < 2.5$.

For each of these levels, Table 9 shows the configuration with the best DR result (average over all scenarios) with their FA average for the case with TI and without TI, respectively. We observe again that the TI configurations achieve better DR values for all five levels. As observed before, algorithm AE supports the lowest FA values for using thread information followed by MLP while the STIDE and BOSC algorithms achieve better DR values when FA levels 1 and 2 are still manageable.

8 Conclusion

Summary – We made it our task to check whether thread information is helpful to improve anomaly-based HIDS. In the search for a suitable data set, we

Table 9. Top configurations for level 1 to 5 with and without TI.

	with TI							without TI						
	alg.	enc.	TI	n	l	$\overline{DR}$	$\overline{FA}$	alg.	enc.	TI	n	l	$\overline{DR}$	$\overline{FA}$
1	STIDE	-	+	5	100	0.983	23.7	BOSC	-	-	7	100	0.710	14.9
2	BOSC	-	+	3	10	0.982	18.9	BOSC	-	-	7	100	0.710	14.9
3	MLP	OHE	+	7	1	0.788	4.2	AE	W 2	-	3	1	0.624	8.2
4	MLP	OHE	+	7	1	0.788	4.2	MLP	W 5	-	3	1	0.528	3.1
5	AE	W 5	+	7	1	0.622	2.2	MLP	W 2	-	3	1	0.526	2.3

ended up with the LID-DS. We then evaluated 7 algorithms on the LID-DS. For each of these algorithms, many different configurations (n-gram length, streaming window length, encoding of the input data, and thread information) were experimentally tested. Particular emphasis has been placed on the difference between the variants with and without thread information. As a result, it can be said that for most cases the use of thread information increases the detection rate and reduces the number of false alarms. In addition, we performed a thought experiment in which different levels of error rates were defined as acceptable. For each of these levels, the best solution was an algorithm that uses thread information. For comparison, we additionally searched for the best matching algorithm without thread information for each of these levels. The results were worse in almost every respect.

Outlook and Open Questions – In future work, we plan to investigate additional algorithms and how they can make use of thread information, in particular based on self organizing maps (SOMs), long short-term memory networks (LSTMs) and the Transformer [32] architecture.

Given that several algorithms have complementary strengths and limitations regarding different scenarios and regarding DR and FA results we also want to investigate whether combined approaches can perform better than single algorithms.

References

1. Abed, A.S., Clancy, C., Levy, D.S.: Intrusion detection system for applications using linux containers. In: International Workshop on Security and Trust Management. pp. 123–135. Springer (2015)
2. Accenture: Securing the digital economy (2019), https://www.accenture.com/gb-en/insights/cybersecurity/_acnmedia/Thought-Leadership-Assets/PDF/Accenture-Securing-the-Digital-Economy-Reinventing-the-Internet-for-Trust.pdf
3. Australian Center for Cyber Security (ACCS): The adfa intrusion detection datasets (2013), https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/
4. Computer Science Department Farris Engineering Center; University of New Mexico: Computer immune systems - data sets and software (1999), https://www.cs.unm.edu/ immsec/systemcalls.htm

5. Creech, G., Hu, J.: Generation of a new ids test dataset: Time to retire the kdd collection. In: 2013 IEEE Wireless Communications and Networking Conference (WCNC). pp. 4487–4492. IEEE (2013)
6. Debar, H., Dacier, M., Wespi, A.: Towards a taxonomy of intrusion-detection systems. Computer networks **31**(8), 805–822 (1999)
7. Eskin, E., Lee, W., Stolfo, S.J.: Modeling system calls for intrusion detection with dynamic window sizes. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01. vol. 1, pp. 165–175. IEEE (2001)
8. European Union: Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (2016), https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504
9. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for unix processes. In: Proceedings 1996 IEEE Symposium on Security and Privacy. pp. 120–128. IEEE (1996)
10. Grimmer, M., Röhling, M.M., Kreusel, D., Ganz, S.: A modern and sophisticated host based intrusion detection data set. IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung pp. 135–145 (2019)
11. Grimmer, M., Röhling, M.M., Kricke, M., Franczyk, B., Rahm, E.: Intrusion detection on system call graphs. Sicherheit in vernetzten Systemen, pages G1–G18 (2018)
12. Haider, W., Hu, J., Slay, J., Turnbull, B.P., Xie, Y.: Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. Journal of Network and Computer Applications **87**, 185–192 (2017)
13. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. Journal of computer security **6**(3), 151–180 (1998)
14. International Data Group: CSO: 2018 u.s. state of cybercrime (2018), https://www.idg.com/tools-for-marketers/2018-u-s-state-of-cybercrime/
15. Jewell, B., Beaver, J.: Host-based data exfiltration detection via system call sequences. In: ICIW2011-Proceedings of the 6th International Conference on Information Warfare and Secuirty: ICIW. p. 134. Academic Conferences Limited (2011)
16. Kang, D.K., Fuller, D., Honavar, V.: Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In: Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. pp. 118–125. IEEE (2005)
17. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
18. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE journal **37**(2), 233–243 (1991)
19. Kruegel, C., Mutz, D., Valeur, F., Vigna, G.: On the detection of anomalous system call arguments. In: European Symposium on Research in Computer Security. pp. 326–343. Springer (2003)
20. Lincoln Laboratory MIT: Darpa intrusion detection evaluation data set (1998-2000), https://www.ll.mit.edu/r-d/datasets
21. Maggi, F., Matteucci, M., Zanero, S.: Detecting intrusions through system call sequence and argument analysis. IEEE Transactions on Dependable and Secure Computing **7**(4), 381–395 (2008)
22. Marceau, C.: Characterizing the behavior of a program using multiple-length n-grams. In: Proceedings of the 2000 workshop on New security paradigms. pp. 101–110 (2001)

23. Mikolov, T., Chen, K., Corrado, G., Dean, J.: word2vec - tools for computing distributed representation of words, https://github.com/tmikolov/word2vec, https://github.com/tmikolov/word2vec
24. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
25. Milenkoski, A., Vieira, M., Kounev, S., Avritzer, A., Payne, B.: Evaluating computer intrusion detection systems: A survey of common practices. ACM Computing Surveys **48**, 12:1– (09 2015). https://doi.org/10.1145/2808691
26. MITRE: Common weakness enumeration - a community-developed list of software & hardware weakness types, https://cwe.mitre.org/, https://cwe.mitre.org/
27. MITRE: Cve - common vulnerabilities and exposures, https://cve.mitre.org/, https://cve.mitre.org/
28. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous system call detection. ACM Transactions on Information and System Security (TISSEC) **9**(1), 61–93 (2006)
29. Pendleton, M., Xu, S.: A dataset generator for next generation system call host intrusion detection systems. In: MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM). pp. 231–236. IEEE (2017)
30. Röhling, M.M., Grimmer, M., Kreubel, D., Hoffmann, J., Franczyk, B.: Standardized container virtualization approach for collecting host intrusion detection data. In: 2019 Federated Conference on Computer Science and Information Systems (FedCSIS). pp. 459–463. IEEE (2019)
31. Schölkopf, B., Williamson, R.C., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. Advances in neural information processing systems **12**, 582–588 (1999)
32. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017)
33. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344). pp. 133–145. IEEE (1999)
34. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344). pp. 133–145. IEEE (1999)
35. Xie, M., Hu, J., Yu, X., Chang, E.: Evaluating host-based anomaly detection systems: Application of the frequency-based algorithms to adfa-ld. In: International Conference on Network and System Security. pp. 542–549. Springer (2015)
36. Zhao, Y., Chu, S., Zhou, Y., Tu, K.: Sequence prediction using neural network classiers. In: International conference on grammatical inference. pp. 164–169 (2017)