Sequence Covering for Efficient Host-Based Intrusion Detection

Pierre-François Marteau

Sequence Covering for Efficient Host-Based Intrusion Detection

Pierre-Francois Marteau

Abstract—This paper introduces a new similarity measure, the covering similarity, that we formally define for evaluating the similarity between a symbolic sequence and a set of symbolic sequences. A pair-wise similarity can also be directly derived from the covering similarity to compare two symbolic sequences. An efficient implementation to compute the covering similarity is proposed that uses a suffix-tree data-structure, but other implementations, based on suffix-array for instance, are possible and possibly necessary for handling very large-scale problems. We have used this similarity to isolate attack sequences from normal sequences in the scope of Host-based Intrusion Detection. We have assessed the covering similarity on two well-known benchmarks in the field. In view of the results reported on these two datasets for the state of the art methods, and according to the comparative study we have carried out based on three challenging similarity measures commonly used for string processing or in bioinformatics, we show that the covering similarity is particularly relevant to address the detection of anomalies in sequences of system calls.

Index Terms—Sequence Covering Similarity, Host-based Intrusion Detection, System Calls, Semi-Supervised Learning, Zero-Day

1 Introduction

Intrusion Detection Systems (IDS) are more and more heavily challenged by intrusion *scenarios* developed by today's hackers. The number of reported intrusion incidents has dramatically increased during the last few years with very serious consequences for organizations, companies and individuals. As an example, the Troyan horse TINBA (which stands for TINy BAnker) has targeted with apparent success the worldwide banking system during the last three years [1], [2], [3]. The detection of zero-day attacks (attacks that have never been detected before) is even more challenging since no pattern or signature characterizing this kind of attack can be used to identify it. Furthermore, with the development of the IoT, the rate of the production of sequences of system calls, i.e. sequential data used to access, manage, or administrate connected equipments, is exploding. Hence, the need to develop and use efficient intrusion detection algorithms that can identify, isolate and handle suspicious patterns in sequential information flows is evermore pressing with time.

This paper addresses the detection of (unknown) anomalies in symbolic sequential data with a specific focus on sequences of system calls within the scope of intrusion detection. A system call is a request to the kernel of an operating system to provide a service on behalf of the user's program. System calls are used to manage the file system and available hardware, control processes, and to provide interprocess communication. Thus, a sequence of system calls corresponds to the sequential list of service requests sent by a process to the kernel, and as such, it constitutes a trace that describes the behavior of the monitored process.

The detection of abnormal system call sequences is challenging for the following reasons:

1) the anomalies are contextual: the occurrence of a system call can be considered as abnormal given its context of occurrence, basically given the subsequences of system calls that occur before and after it,
2) the anomalies can be collective: a subsequence as a whole can be considered as abnormal,
3) the variability of system call sequences is very high mainly because they are of varying length (a low number to a few thousand system calls) and the alphabet on which the system calls are defined is quite high (more than 300 for the Linux system),
4) if subsequences can be seen as discriminative features, the combinatorics increases dramatically with the size of the subsequences. This prohibits the use of n-gram features for example, when n is above a low number of system calls (4, 5).

Bioinformatics has proposed over the years a great number of similarity measures [4], [5], [6] to compare pairs of sequences or provide multiple sequence alignments for sets of sequences. These editing distances cover partially the above-mentioned difficulties and have been used with great success to evaluate similarities and dissimilarities in sequences of nucleotides, some of them allowing for the isolation of abnormal subsequences [7]. In addition, they have been successfully adapted to cope with sequences of system calls [8]. However, the quadratic time complexity of most of these similarity measures and the need for the computation of the whole similarity matrix limit their use to small to medium size problems. Approximate computations of these measures along with extremely parallel hardware are required to address large-scale problems [9].

In this context, designing an algorithm that can somehow profit from subsequences of any size to evaluate the pairwise similarity of sequences with a significantly lower time complexity is challenging. Furthermore, in the scope of anomaly-based intrusion detection, if this similarity mea-

• *P.-F.. Marteau was with the Institut de Recherche en Informatique et Systemes Aleatoires (IRISA) Laboratory, Universite Bretagne Sud, France. E-mail: https://people.irisa.fr/Pierre-Francois.Marteau/*

Manuscript received xxx, 2017; revised yyy.

sure can be used to compare a test sequence to a set of normal sequences, it can result in a quite and efficient intrusion detection approach.

Our main contribution is the description of an efficient algorithm (SC4ID, which stands for Sequence Covering for Intrusion Detection) based on the concept of a so-called *optimal-covering* of a sequence by a series of subsequences extracted from a predefined set of sequences. We demonstrate in this paper that this algorithm is efficient, at least in terms of accuracy and response time to isolate attack sequences that have never been observed (zero-day settings). Indeed, as its implementation is based on generalized suffix-trees (or suffix-arrays), it requires a relatively large memory overhead.

The second section of this paper briefly reports the related key works. We detail the SC4ID algorithm in the third section and evaluate it in the fourth section on two distinct sequences of system call benchmarks. On the smallest size benchmark we compare the proposed similarity measure with three baseline similarity measures commonly used in bioinformatics to estimate the similarity of symbolic sequences. We discuss our results in the fifth section before concluding this study in section six.

2 Context and Related works

In a broad sense we address anomaly detection in sequential data [10] while focusing on intrusion detection in cyber-physical systems. Intrusion [11] refers to possible security breaches in (cyber-)systems, namely malicious activity or policy violations. It covers both intrusions *per se*, i.e. attacks from the outside, and misuse, i.e. attacks from within the system. An intrusion detection system (IDS) is thus a device that monitors a system for detecting potential intrusions. The IDS will be referred to as NIDS if the detection takes place on a network and HIDS if it takes place on a host of a network. Furthermore, we distinguish i) signature-based IDS approaches, that detects attacks by looking for predefined specific patterns, such as byte sequences in network packets, or known malicious sequences of instructions used by malware, to ii) anomaly-based intrusion detection systems that were primarily introduced to detect unknown attacks (zero-day attacks).

In this work we exclusively address host intrusion detection system (HIDS) through semi-supervised anomaly-based approaches. Since Forrest's pioneering work [12] most of HIDS (at least in the UNIX/LINUX sphere) use system call sequencesas their primary source of information. Generally, sequences of system calls are represented as sequences of integer symbols, for which the order of occurrence of the symbol is of crucial importance. Numerous work and surveys have been published in the area of anomaly detection in the scope of intrusion detection, see [13] [14], [15] for recent studies. If we reduce the area of interest to anomaly detection in sequential data, four avenues for handling symbolic sequences are mainly followed:

Window-based approaches [16] are quite popular since a fixed size window enables a wide range of statistical, knowledge-based and machine learning techniques to be applied in a straightforward manner. A fixed-size window is first defined, and then it progressively slides along the tested sequence. Each window (basically a fixed-size subsequence) is in general represented by a feature vector. Then, models such as the one class Support Vector Machine [17], Multi Layer Perceptron and Convolutional Neural Networks [18] are used to provide a score for deciding whether an anomaly is present or not.

In this framework, an aggregation of each window (that is sliding all along the sequence) score is necessary to get a global decision at the sequence level. The prediction score is used to detect and locate the position of the anomalies inside the test sequence if any.

Global kernel-based approaches [19] process each sequence as a whole and a pair-wise sequence kernel (string kernel) is used to provide the sequence space with a similarity measure. The k-Near-Neighbor rule or any of the so-called kernel machine methods can then be applied to model the 'normal' clusters and isolate the 'anomalies'. These approaches find their roots in text processing [20] (Levenshtein's distance) or in Bioinformatics [4], [21] (Smith and Watermans), [5] (Needleman-Wunsch), [22] (BLAST) and Longest Common Subsequence (LCS) or Longest Similar Subsequence [23] [24]. Such methods do not seem to outperform window-based approaches and are in general much more costly in term of algorithmic complexity than state of the art methods.

Generative approaches, essentially Hidden Markov Models (HMM) [25] [26] [27], Conditional random Fields (CRF) [28] [29] or Recurrent Neural Networks (RNN, LSTM, etc.) [30] [31] have been used with apparent success on various intrusion detection tasks, such as payload analysis or Network Layer Intrusion Detection or HIDS. However, the choice of parameters such as the order of the Markovian dependency, number of hidden variables, etc., is often the result of a compromise to avoid over-fitting, and long-term dependency is not necessarily easily modeled.

Language-based approaches have been proposed initially to extract very simple n-gram features to enhance a vector space model similar to the one used in text mining [32], [33]. Recently, a much ambitious model has been proposed that proposes to enact phrases and sentences, hence a language, from sequences of system calls [34]. Nevertheless, these approaches suffer from the combinatorics explosion. When simple n-grams models are used (with n lower than 5 or 7) the size of the vector space model is very high (several millions of dimension) and in general the lack of available data to train the model limits its accuracy. In the case of Creech et al. approach [34], the combinatorics is much higher with an estimated feature space dimension of 10^{16} which makes this model intractable for common hardware.

The approach we present below relates to the (global) kernel-based family. Each sequence is thus considered as a whole, regardless of its length. All the specificity and novelty of the method relies on a quite simple similarity measure, that we call covering similarity. To our knowledge,

this similarity measure has not been yet proposed for sequence comparison, specifically in the context of intrusion detection. It is defined to evaluate how close a sequence is to a set of supposedly 'normal' sequences. A simple threshold is used to decide whether an unknown sequence is 'normal' or should be considered as an anomaly. The main advantages of our approach are:

- apart from a decision threshold, it is parameter-free, in particular it does not rely on a window size,
- it is incremental and can be set up 'online',
- it is interpretable since it easily enables the location of abnormal areas in long sequences,
- it supports a very efficient instance selection scheme to iteratively improve the 'normal' model, without overloading it with unnecessary instances,
- it is quite efficient compared to other machine learning based models and scales well compared to classical sequence alignment kernels (which, in general, are at least in $O(n^2)$ complexity) such as string kernels, the Longest Common Subsequence or the Smith and Waterman similarities, due to the suffix-tree and suffix-array data structures on which the implementation of our algorithm relies.
- Furthermore, it does not rely on pairwise distance calculation between sequences, but on the distance between a sequence and a set of reference sequences, which also drastically reduces its computational complexity.

3 The SC4ID Algorithm

The overall principle of the SC4ID algorithm is straightforward. It is depicted in algorithm 1 and presented in Fig. 1. Given a set of sequences considered as normal sequences, S, and a threshold $\sigma \in [0; 1]$, SC4ID evaluates the similarity of an unknown sequence s with the elements of S according to a similarity measure $\mathscr{S}(s, S)$ that consists in optimally covering s with subsequences of elements of S. SC4ID can be considered as an implicit semantic-based approach if we interpret the covering subsequences as phrases constructed from S, the corpus of training sequences. SC4ID also relates to editing distances or time elastic measures such as the Levenshtein's distance. But instead of addressing the matching problem through local editing operations (substitution, insertion, deletion), it evaluates the minimal number of subsequences that are required to build a complete covering of a given sequence. Furthermore, it allows for subsequences to be swapped, an operation that is very costly to implement in the editing distance framework. Most importantly, SC4ID does not evaluate pair-wise sequence similarities to construct a whole similarity matrix as for the Levenshtein's distance, but directly computes the similarity between a sequence and a set of sequences which drastically reduces its algorithmic complexity.

Behind this covering principle, we potentially manage to use any subsequence that can be drawn from the set S, without constraining neither the length of the subsequences nor their number. The covering is in itself informative and enables the location of anomalies spread inside a long sequence. Furthermore one can interpret the covering as a set of words that describes the sequence, seen as a phrase

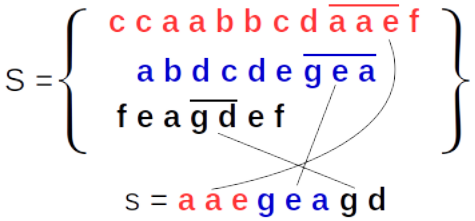


Fig. 1. Principle of SC4ID. In the example, subsequence s is optimally covered by three subsequences drawn from sequences in S, $[aae, gea, gd]$, which is a S-optimal covering for s.

(as conceptually proposed in [35]). Indeed the fact that a fast algorithm exists to evaluate S-optimal coverings makes it very relevant in the context of anomaly detection in sequential data.

Finally, as described in algorithm 1, if the covering similarity is above the threshold σ then the sequence s is considered as normal, otherwise it will be considered as an anomaly.

The only parameter in the SC4ID algorithm is σ. In a semi-supervised anomaly-based detection framework applied to intrusion detection, all methods provide a confidence measure (e.g a probability, a distance, a similarity measure, etc.) and the definition and setting up of an σ-like parameter in order to make the final decision. We discuss the tuning of this parameter for the SC4ID algorithm in subsection 4.3.

The specificity and novelty of the algorithm lie in the way similarity $\mathscr{S}(s, S)$ is defined. We introduce hereinafter some definitions and notation to detail the formal definition of this similarity measure.

3.1 Definitions and notation

Let Σ be a finite alphabet and let Σ^* be the set of all sequences (or strings) defined over Σ. We note ϵ as the empty sequence.

Let $S \subset \Sigma^*$ be any set of sequences, and let S_{sub} be the set of all subsequences that can be extracted from any element of $S \cup \Sigma$. We denote by $\mathscr{M}(S_{sub})$ the set of all the multisets[1] that we can compose from the elements of S_{sub}.

$c \in \mathscr{M}(S_{sub})$ is called a partial covering of sequence $s \in \Sigma^*$ if and only if

1) all the subsequences of c are also subsequences of s,
2) indistinguishable copies of a particular element in c correspond to distinct occurrences of the same subsequence in s.

If $c \in \mathscr{M}(S_{sub})$ entirely covers s, meaning that we can find an arrangement of the elements of c that covers s entirely, then we will call it a full covering for s.

Finally, we call a S-optimal covering of s any full covering of s which is composed with a minimal number of subsequences in S_{sub}.

Let $c^*_S(s)$ be a S-optimal covering of s.

1. A multiset is a collection of elements in which elements are allowed to repeat; it may contain a finite number of indistinguishable copies of a particular element.

We define the covering similarity measure between any non-empty sequence s and any set $S \subset \Sigma^*$ as

$$\mathscr{S}(s, S) = \frac{|s| - |c^*_S(s)| + 1}{|s|} \quad (1)$$

where $|c^*_S(s)|$ is the number of subsequences composing a S-optimal covering of s, and $|s|$ is the length of sequence s.

Note that in general $c^*_S(s)$ is not unique, but since all such coverings have the same cardinality, $|c^*_S(s)|$, $\mathscr{S}(s, S)$ is well defined.

Properties of $\mathscr{S}(s, S)$:

1) If s is a non-empty subsequence in S_{sub}, then $\mathscr{S}(s, S) = 1$ is maximal.
2) In the worst case, the S-optimal covering of s has cardinality equal to $|s|$, meaning that it is composed only with subsequences of length 1. In that case, $\mathscr{S}(s, S) = \frac{1}{|s|}$ is minimal.
3) If s is non-empty, $\mathscr{S}(s, \emptyset) = \frac{1}{|s|}$ (note that if $S = \emptyset$, $S_{sub} = \Sigma$).

Furthermore, as ϵ is a subsequence of any sequence in Σ^*, we define, for any set $S \subset \Sigma^*$, $\mathscr{S}(\epsilon, S) = 1.0$

Note that the covering similarity between a sequence and a set of sequences as defined in Eq. 1 enables the definition of a covering similarity measure on the sequence set itself. For any pair of sequences s_1, s_2 this measure is defined as follows:

$$\mathscr{S}_{seq}(s_1, s_2) = \frac{1}{2}(\mathscr{S}(s_1, \{s_2\}) + \mathscr{S}(s_2, \{s_1\})) \quad (2)$$

where $\mathscr{S}$ is defined in Eq. 1.

Algorithm 1: SC4ID

input : $S \subset \Sigma^*$, a set of sequences
input : $s \in \Sigma^*$, a test sequence
input : $\sigma \in [0, 1]$, a threshold value
output: a decision value: 'normal' or 'anomaly'

Provide a S-optimal covering of s;
Evaluates $\mathscr{S}(s, S)$ according to Eq. 1;
if $\mathscr{S}(s, S) \geq \sigma$ **then return** ['normal', $\mathscr{S}(s, S)$];
else return ['anomaly', $\mathscr{S}(s, S)$];

As an example, let us consider the following case:

s_1 = [0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1]
s_2 = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1]
$S = \{s_1, s_2\}$
s_3 = [0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1]
s_4 = [0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]

The S-optimal covering of s_3 [2] is size 4, hence $\mathscr{S}(s_3, S) = \frac{16-4+1}{16} = 13/16$, and the S-optimal covering of s_4 [3] is size 8, leading to $\mathscr{S}(s_4, S) = \frac{16-8+1}{16} = 9/16$.

The main challenge for the SC4ID algorithm is to evaluate $\mathscr{S}(s, S)$ efficiently for a sufficiently large S and relatively long sequences s such as to be able to process common sequences of system calls. This essentially requires an efficient way to get S-optimal coverings for tuples (s, S) constructed from general sequences of system calls.

2. ([0,0,1,1][0,0,1,1],[0,0,1,1][0,0,1,1]) is a S-optimal covering of s_3
3. ([0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1]) is a S-optimal covering for s_4

3.2 Finding a S-optimal covering for any tuple (s, S)

The brute-force approach to find a S-optimal covering for a sequence s is presented in algorithm 2. It is an incremental algorithm that, first, finds the longest subsequence of s that is contained in S_{sub} and that starts at the beginning of s. This first subsequence is the first element of the S-optimal covering. Then, it searches for the next longest subsequence that is in S_{sub} and which starts at the end of the first element of the covering, then adds it to the covering under construction, and it is then iterated until reaching the end of sequence s.

Algorithm 2: Find a S-optimal covering for s

input : $S \subset \Sigma^*$, a set of sequences
input : $s \in \Sigma^*$, a test sequence
output: c, a (S-optimal) covering for s

$continue \longleftarrow True$;
$start \longleftarrow 0$;
$c^* \longleftarrow \emptyset$;
while *continue* **do**
  $end \longleftarrow start + 1$;
  while $end < |s|$ *and* $s[start : end] \in S_{sub}$ **do**
    $end \longleftarrow end + 1$;
  $c \longleftarrow c^* \cup \{s[start : end - 1]\}$;
  if $end = |s|$ **then** $continue \longleftarrow False$;
  $start \longleftarrow end$;
return c;

Proposition 3.1. *Algorithm 2 outputs a S-optimal covering for sequence s.*

Proof. i) First we observe that since all the subsequences of length 1 constructed on Σ are included into S_{sub}, algorithm 2, by construction, outputs a full covering of s (meaning that s is entirely covered by the subsequences of the covering provided by the algorithm).

ii) Second we observe that, for all s_1 and s_2 in Σ^* such that s_1 is a subsequence of s_2, and any $S \subset \Sigma^*$, $|c^*_S(s_1)| \leq |c^*_S(s_2)|$.

We finalize the proof by induction on n, the cardinality (the size) of the coverings.

The proposition is obviously true for $n = 1$: for all sequence s for which a covering of size 1 exists (meaning that s is a subsequence of one of the sequences in S), algorithm 2 finds the S-optimal covering that consists of s itself.

Then, assuming that the proposition holds for n, such that $n \geq 1$ (IH), we consider a sequence s that admits a S-optimal covering of size $n + 1$.

Let $s = s_1 + \overline{s}_1$, be the decomposition of s according to the full covering provided by algorithm 2, where s_1 is the prefix of the covering (first element) and $\overline{s}_1$ the remaining suffix subsequence (concatenation of the remaining covering elements). $+$ is the sequence concatenation operator. Similarly, Let $s = s^*_1 + \overline{s}^*_1$, be the decomposition of s according

to a S-optimal covering of s. Essentially, s_1^*, which is also a prefix of s, is a subsequence of s_1 (otherwise, since s_1^* is in S_{sub}, algorithm 2 would have increased the length of s_1 at least to the length of s_1^*). Hence, $\overline{s}_1$ is a subsequence of $\overline{s}_1^*$ and, according to ii), $|c_S^*(\overline{s}_1)| \leq |c_S^*(\overline{s}_1^*)| = n$. This shows that $\overline{s}_1$ is a sequence that admits a S-optimal covering, $c_S^*(\overline{s}_1)$, of a size at most equal to n. According to (HI), algorithm 2 returns such an optimal covering for $\overline{s}_1$. This shows that the covering $\{s_1\} \cup c_S^*(\overline{s}_1)$ that is returned by algorithm 2 for the full sequence s, is at most size $n+1$, meaning that it is actually a S-optimal covering for s of size $n+1$. Hence, by induction, the proposition is true for all n, which proves the proposition. □

Algorithm 3: Find the first break location in s between positions t_b and t_e

```
Function breakDichoSearch(s, t_b, t_e, S)
    input : s ∈ Σ*, a test sequence
    input : t_b < t_e < |s|, the index segment in which
            looking for the break
    input : S ⊂ Σ*, a set of sequences
    output: t, the searched breaking index position
    t ⟵ ⌊(t_b + t_e)/2⌋;
    if t = t_b and s[t_b : t_e] ∈ S_sub then
        return t+1
    else
        return t
    if s[t_b : t] ∈ S_sub then
        return breakDichoSearch(s, t, t_e, S);
    else
        return breakDichoSearch(s, t_b, t, S);
```

Algorithm 4: Find using a binary search a S-optimal covering for s

```
input : S ⊂ Σ*, a set of sequences
input : s ∈ Σ*, a test sequence
output: c*, a S-optimal covering for s
continue ⟵ True;
start ⟵ 0;
c* ⟵ ∅;
while continue do
    t ⟵ breakDichoSearch(s, start, |s|, S_sub);
    c* ⟵ c* ∪ {s[start : t − 1]};
    if t = |s| then continue ⟵ False;
    start ⟵ t;
return c*;
```

3.3 Algorithmic complexity and implementation considerations

Algorithm 2 requires a fast way to test the existence of a subsequence in the sequences of S. Similarly to many algorithms in the field of information indexing and retrieval, these algorithms face two difficulties when the size of the data (the size of S and the average length of the sequences) increases, namely memory consumption and response time. These two aspects cannot be solved simultaneously and require a compromise to be found. We discuss below some potential implementations based on hashtable, suffix-tree or suffix-array.

3.3.1 Hashtable implementation for the search of a subsequence

If response time is ideal, and memory space is not a problem, then one can implement a hashtable that stores all the subsequences of S (more precisely the elements of S_{sub}). By doing so, we will be able to know if a subsequence is a member of S_{sub} in $O(1)$ time complexity. Hence, the time complexity to find the S-optimal covering for a sequence s of average size n will be $O(n)$ for this implementation.

On the other hand, if the average length of the sequences in S is n, then, the space required to store all the elements of S_{sub} is expressed in $O(n^2 \cdot |S|)$.

This could be feasible for small size problems. However, for long sequences, e.g. with an average length of 10^5 elements, and large S, e.g. 10^6 sequences, we need to store $O(10^{16})$ subsequences in the hashtable which is not feasible in practice on common hardware.

3.3.2 Suffix-tree and suffix-array implementations for the search of a subsequence

For medium to large size problems, we need to drastically limit the space consumption.

In comparison to a hashtable implementation, a generalized suffix-tree implementation [36] [37] would reduce the memory requirement to $O(n \cdot |S|)$, although, in general, with a large proportionality constant (typically 10 to 100 in practice), and a slight increase of the computational complexity for searching for a subsequence ($O(m)$, where m is the length of the subsequence that is searched).

In comparison, suffix-arrays [38], and specifically its enhanced implementation [39], is a space-efficient data-structure that reduces the memory consumption without losing (too much) on the response time. As they are cache friendly, a suffix-array can in practice enable handling much larger sequence sets than a suffix-tree and is much easier to parallelize. A suffix-array provides the search for a subsequence with $O(m + log(n \cdot |S|))$ average time complexity (where n is the average length of the sequence in S).

3.3.3 Overall time complexity

In our current implementation, we gave priority to speed rather than memory consumption, while trying to run our algorithm on common hardware. Thus we adopted a generalized suffix-tree implementation in Python[4] that we have modified to cope with sets of sequences of integers (each integer corresponding to a system call) instead of sets of strings. The sequence datasets, described below, that we have used for our experiment fit easily in a generalized suffix-tree. Hence, using a generalized suffix-tree implementation, the search for a subsequence of size n is $O(n)$.

4. Python implementation of Suffix Trees and Generalized Suffix Trees, https://github.com/ptrus/suffix-trees

The main computing effort for algorithm 2 is located in the second part of the test (at line 6), which consists of checking whether the subsequence $s[start : end]$ belongs to the set of subsequences S_{sub} associated to S. As we have to iterate along the sequence s to successively find the elements of its covering, algorithm 2 would require searching $O(|s|)$ subsequences, leading to an $O(|s|^2)$ upper bound for the total time complexity.

Indeed, a straightforward improvement of the brute-force algorithm can be achieved. Instead of iterating along the sequence s to find successively the elements of its covering, this improvement implements a dichotomic (or binary) search to locate the extremities, that we call breaks, of the subsequences that compose the covering. This improvement is described in algorithms 3 and 4.

The improved search algorithm has a computational complexity that is upper bounded by $O(k \cdot |s| \cdot log(|s|))$, where $k = |c^*_S(s)|$ is the size of a S-optimal covering for s. Note that, if the size, k, of the covering is of the same order of magnitude as the length of s, then the 'improved' algorithm would, in fact, require more time than the brute-force one. Hence the improvement is only achieved when the size of the covering is significantly smaller than the length of the covered subsequence, which is the case in general, except for *abnormal* sequences that we are aiming to isolate. Such sequences are expected to be rare, and, in principle, we could accept the extra computing cost if we effectively manage to separate them from the flow of numerous *normal* sequences that need to be processed.

In addition the previous time complexities for Algorithms 2 and 1 do not depend on $|S|$, which means that increasing the size of S will not impact on the processing time. This property is particularly important for applications for which $|S|$ is potentially large such as in anomaly-based intrusion detection for instance.

4 EXPERIMENT

	UNM	ADFA-LD
$\|S\|$	81	5951
min seq. length	7	75
max seq. length	183018	4494
mean seq. length	9031	462
std seq. length	25111	522

TABLE 1
Some statistics on the UNM and ADFA-LD datasets.

We have evaluated the SC4ID algorithm on two well-known system call datasets provided respectively by the University of New Mexico (UNM) [40] and the Australian Centre for Cyber-Security, referred to as ADFA-LD [35]. Some statistics summarizing the content of these datasets are given in Table 1. We have selected these two datasets because i) their use by the research community in intrusion detection covers a wide time span (1998-2013) and ii) numerous results have been reported using these datasets in various settings, which enable to compare our findings to the state of the art.

We have used as baselines, three similarity measures commonly used in text processing or bioinformatics, namely the Levenshtein's distance (LEV) [20], [41], the Longest Common Subsequence (LCSq) [42], [43] and the Longest Common Substring (LCSt) [44].

The longest common substring problem, also known as the longest common contiguous subsequence problem, is to find the longest string that is a substring of two or more strings. The pairwise similarity based on the longest common substring that we used is normalized as follows :

$$\mathcal{S}_{LCSt}(s_1, s_2) = \frac{LCSt(s1, s2)}{max(|s1|, |s2|)} \tag{3}$$

where $LCSt(s1, s2)$ is the length of the longest common substring of s_1 and s_2. An implementation using suffix-trees leads to a $O(n + m)$ algorithmic complexity, where m and n are the lengths of the pair of strings or sequences that are compared.

The Longest Common Subsequence is similar to a longest common substring problem, except that the symbols of the longest subsequences do not need to be contiguous in the input sequences, namely gaps are authorized. The pairwise similarity based on the longest common subsequence that we used is normalized as follows :

$$\mathcal{S}_{LCSq}(s_1, s_2) = \frac{LCSq(s1, s2)}{max(|s1|, |s2|)} \tag{4}$$

where $LCSq(s1, s2)$ is the length of a longest common subsequence of s_1 and s_2. Using a dynamic programming implementation, the algorithmic complexity of LCSq distance is $O(nm)$, where m and n are the lengths of the pair of strings or sequences that are compared. The LCSq distance is conceptually close to the Smith and Waterman [4] distance used in Bioinformatics.

The Levenshtein's distance, also known as the edit distance, is the minimum number of single-symbolic edit operations (insertions, deletions or substitutions) required to change one string or sequence into the other. Using a dynamic programming implementation, the algorithmic complexity of the Levenshtein's distance is $O(nm)$, where m and n are the lengths of the pairs of strings or sequences that are compared. The Levenshtein's distance is conceptually close to the Needleman Wunsch [5] distance used in Bioinformatics.
The pairwise similarity based on the Levenshtein's distance that we used is normalized as follows :

$$\mathcal{S}_{LEV}(s_1, s_2) = 1 - \frac{LEV(s1, s2)}{max(|s1| + |s2|)} \tag{5}$$

where $LEV(s1, s2)$ is the Levenshtein's distance between s_1 and s_2.

The averaged costs for the computation of the similarity matrix required to solve the HIDS problem are

- $O(k \cdot n \cdot logn)$ for $\mathcal{S}(s1, S)$
- $O(N \cdot n)$ for $\mathcal{S}_{LCSt}$
- $O(N \cdot n^2)$ for $\mathcal{S}_{LCSq}$ and $\mathcal{S}_{LEV}$

where N is the size of the normal sequences used as training, n is the average length of the training and test sequences and k is the average length of the S-optimal coverings of the test sequences.

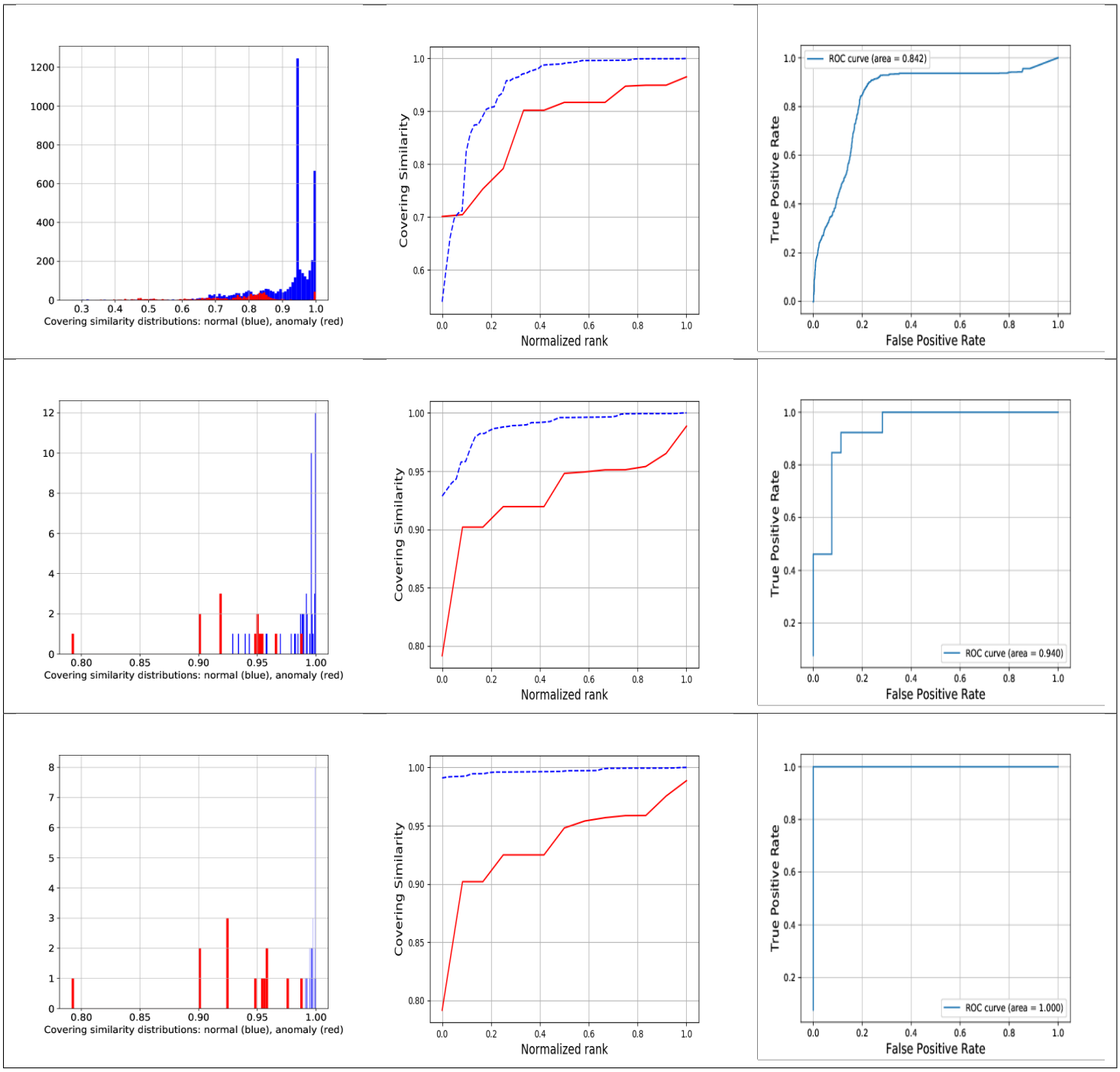


Fig. 2. UNM dataset: histogram of the covering similarity distributions $\mathscr{S}_c(s, S)$ (left column), ranked covering similarity $\mathscr{S}_c(s, S)$ (middle column), ROC curves (right column), when 10% (top row), 22% (middle row) and 38% (bottom row) of the normal data is used for training.

To assess the experimented algorithms on the binary classification task that consists of separating attacks from normal data, we use the *receiver operating characteristic* (ROC) curve that shows the detection rate (true positive rate) as a function of the false alarm rate (false positive rate) as the discrimination threshold of the classifier is varied. We use the *Area Under the ROC curve* (AUC) as the global assessment measure.

4.1 The UNM dataset

For our first experiment, we have considered the sendmail system call traces from the relatively old UNM dataset [16]. These synthetic data were collected at UNM on Sun SPARC-stations running unpatched SunOS 4.1.1 and 4.1.4 with the included sendmail. We have adopted the setting described in [45], basically 68 unique process traces in the normal dataset (uniqueness is to ensure that no sequence appears simultaneously in the training and validation subsets) and 13 process traces in the abnormal dataset (we have kept the 3 duplicated attack sequences in this set).

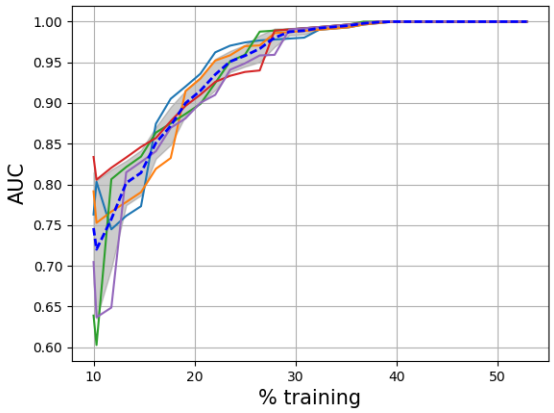


Fig. 3. UNM dataset: Area Under the ROC Curve (AUC) curves as a function of the size of the training set as a percentage for 5 distinct runs. The average curve is shown as a dotted line. The grey filled area shows the +1/-1 standard deviation curves.

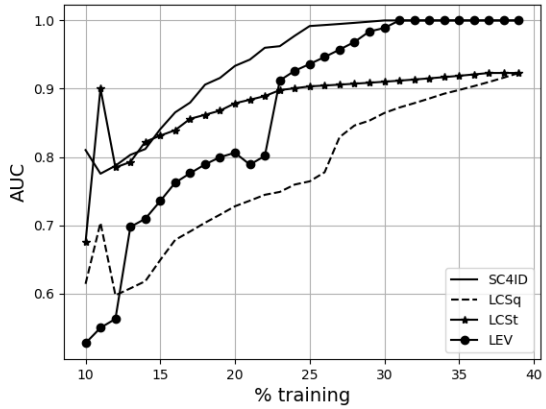


Fig. 4. UNM dataset: AUC curves as a function of the size of the training set as a percentage for SC4ID (plain curve), LEV (circle), LCSt (star), and LCSq (dotted line) similarity measures.

The experimental protocol for this data set and the covering similarity is as follows

1) Initialization: randomly select 10% of the normal data to build the 'normal' model, i.e. the initial set of normal data, S, from which the tested similarity will be evaluated.
2) Evaluation: for each of the remaining normal and attack sequences, s , evaluate the covering similarity $\mathscr{S}(s, S)$. Then rank the normal data according to their similarity score. Finally evaluate the ROC curve, and assessment measure (AUC).
3) Selection: from the previously ranked normal data, select the normal sequence with the worst similarity score, s^-, and update the normal model with training set $S := S \cup \{s^-\}$. Then loop in step 2 until 50% of the normal data is used in training.

The same protocol is used for the other tested similarities, except that the normalized distance to the closest 'normal' sequence is used instead of $\mathscr{S}(s, S)$.

Fig. 2 presents the histogram of the covering similarity values (left column) for the 'normal' (blue) and 'attack' (red) data, the ordered similarity values in increasing order (middle column) for the 'normal' (blue dotted line) and attack data (red, continuous line) and the ROC curve (right column). In this figure, the top row corresponds to a situation for which 10% of the normal data has been randomly selected for training, the middle row where the initial 10% of training data has been enriched using 12% of the remaining normal data corresponding to the lowest covering similarity scores, and finally, the bottom row corresponds to an enrichment of the initial 10% of training data using 28% of the remaining normal data. From top to bottom, we show that the model improves its capacity to separate attack data from normal data: the AUC value that is initially $.81$, reaches $.94$ when 22% of the 'normal' data is used as training and 1.0 when 38% of the 'normal' data is used as training. In the middle column, we see that the similarity scores for the normal data is progressively tangent to the 1 constant curve (maximal similarity value), while, for the attack data, it generally stays much lower, although it increases slightly when the size of the training set increases.

Figure 3 presents the AUC value obtained by the SC4ID algorithm as the enrichment of the training data increases. 5 different runs have been carried out, each one being initialized by randomly selecting 10% of the normal data. The grey area corresponds to the $+1/-1$ standard deviation, and the blue dotted line is the average curve. We can see on this figure that the SC4ID algorithm improves quite rapidly until reaching a perfect separation of the normal and attack data when 38% of the normal data is used, regardless of the initialization. This is a major and very promising result, since the instance selection that is performed directly from the covering similarity scores is working particularly well on the UNM data.

Figure 4 compares the 4 tested algorithms on a single run . In the Figure, one can see that the SC4ID algorithm (plain curve) is the first to reach a perfect AUC value when about 30% of the available normal data is used as training. The Levenshtein similarity based algorithm (LEV) is the second to reach a perfect AUC value when about 31% of the available normal data is used as training data. The LCSq algorithm reaches an asymptotic AUC value of $.923$ and is unable to separate normal data from attack data. Due to the presence of gaps in the longest common subsequences that are extracted, some attacks become too similar to normal data and perfect separation is no longer possible. We deduce from this result that the Smith and Waterman distance used in bioinformatics will not perform well either in this classification task. Finally, the LCSt distance will reach a perfect AUC value when 57% of the available normal data is used as training data, which is almost twice the size of training data than it was necessary for SC4ID or LEV.

Table 2 gives the average elapsed time for each enrichment iteration. We can clearly see that SC4ID is the faster

Similarity	Elapsed time (sec.)
SC4ID	454
LCSt	653
LCSq(*)	27023
LEV(*)	37296

TABLE 2
Elapsed time for the four tested algorithms. SC4ID: Covering Similarity, LCSt: Longest Common Substring, LCSq: Longest (non contiguous) Common Subsequence, LEV: Levenshtein. (*) note that for LCSq and LEV, 20 processor cores have been used.

algorithm, LCSt takes 44% more time and LCSq and LEV take more than 6000% more time than SC4ID. Note that LEV and LCSq were running on a 20 cores architecture while SC4ID and LCSt have not been parallelized, due to their suffix-tree implementation, and were running on a single core.

4.2 The ADFA-LD dataset

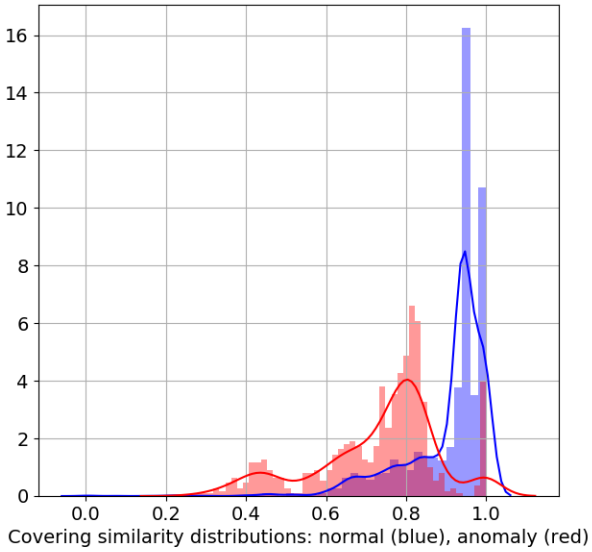
The second experiment involves a much more recent benchmark dataset for HIDS assessment. Yhis benchmark has been designed by the Australian Centre OF Cyber-Security (ACCS). According to the authors, the ADFA-LD dataset [34], [35] has been designed to reflect modern hacking techniques, whilst using modern patched software as its backbone. It is composed with a training set composed with 833 normal sequences, a validation set composed with 4373 normal sequences and a set of 746 attack sequences partitioned into 6 categories referred to as HydraFTP, Hydra-SSH, Adduser, Java-Meterpreter, Meter-preter and Webshell.

Similarly to the UNM experiment, the experimental protocol for this data set is as follows :

1) Initialization: select the 833 sequences of the normal training data to build the 'normal' model, i.e. the initial set of normal data, S, against which the covering similarity will be evaluated.
2) Evaluation: for all the remaining normal and all attack sequences, s, evaluate the covering similarity $\mathscr{S}(s, S)$. Then rank the normal data according to their similarity scores. Finally evaluate the ROC curve and assessment metrics.
3) From the previously ranked normal data, select 100 sequences of normal data with the worst similarity score, S_{100}, update the normal model $S := S \cup S_{100}$ and loop in step 2 until 50% of the normal data is used in training.

When only 833 normal training sequences are used to train SC4ID, one acquires the histogram of similarity scores given in the top of Fig. 5. A red peak of maximal similarity (1.0) exists, meaning that some attack data (actually 32 sequences) are exact subsequences of the 833 normal training data. When we remove these 32 attack sequences, one observes a small remaining red peak with high covering similarity close to 1.0. This peak corresponds to 11 attack sequences that have a covering of size 2, i.e. that are composed of two subsequences that belong to the set of training data.

This situation for which attack sequences can be exact subsequences of normal data is a bit surprising. We do not have to investigate the nature of such subsequences. Rather,



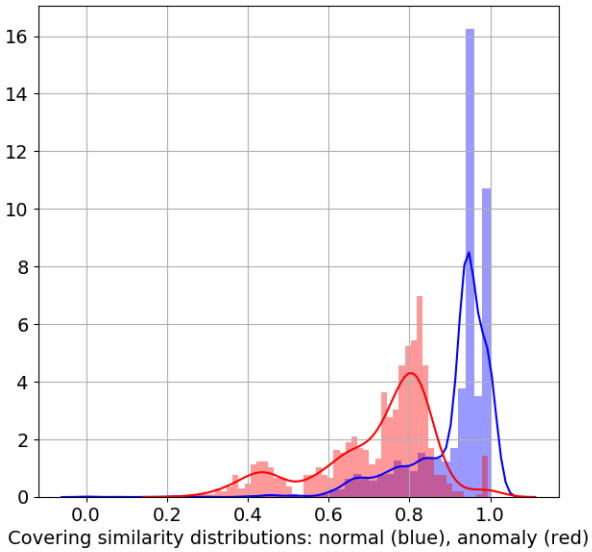


Fig. 5. Histogram of the covering similarity for normal validation data (blue), and attack data (red). Top: all the attacks are considered. Notice the red pick with maximal similarity, 1.0, corresponding to attacks that are exact subsequences of the train sequences. Bottom: the same histogram when the 32 attacks with covering similarity equal to 1.0 are removed. Notice the remaining small peak of high similarity close to 1.0

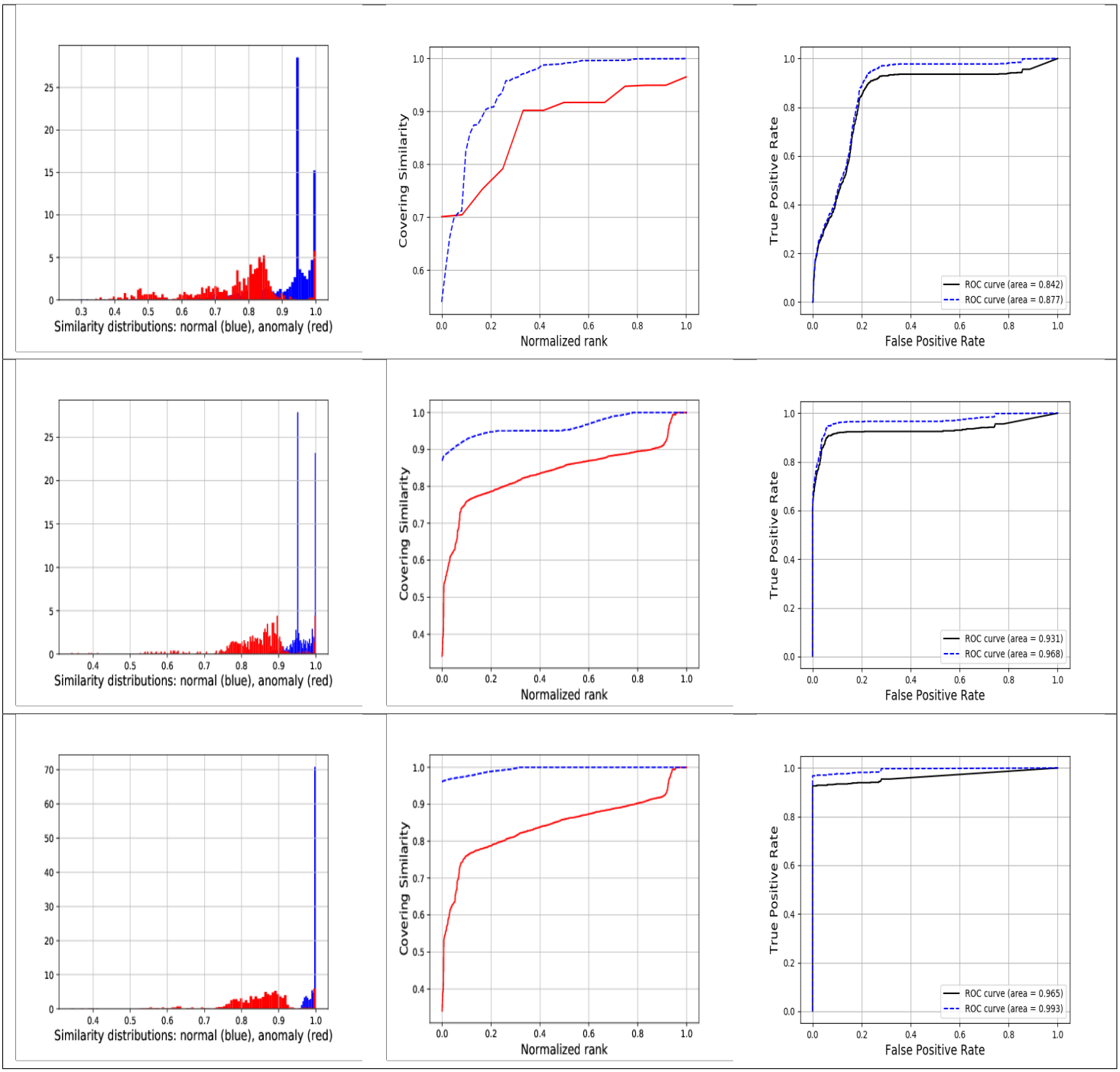


Fig. 6. ADFA-LD dataset: histograms of the covering similarity distributions $\mathscr{S}_c(s, S)$ (left column), ranked normalized covering similarities $\mathscr{S}_c(s, S)$ (middle column), ROC curves (right column), when 833 normal data (top row), 833 + 500 normal data (middle row) and 833 + 1000 normal data (bottom row) is used for training.

we take the opportunity to highlight an obvious limitation of SC4ID: it is completely blind to this kind of situation. Complementary approaches are required to specifically address this issue if any such situation exists.

Fig. 6 presents the histogram of the covering similarity values (left column) for the 'normal' (blue) and 'attack' (red) data, the ordered similarity values in increasing order (middle column) for the 'normal' (blue dotted line) and attack data (red, continuous line) and two ROC curves (right column): the black continuous line curve corresponds to the situation where all the attack data is retained, the blue dotted line corresponds to the situation where the 32 attack sequences that are exact subsequences of the 833 normal training data have been removed. In this figure, the top row corresponds to the situation where only the 833 normal training sequences are used for training, the middle row where the initial 833 sequences have been enriched using 500 sequences of the remaining normal data that have the lowest covering similarity, and finally the bottom row corresponds to a an enrichment of the initial 833 training

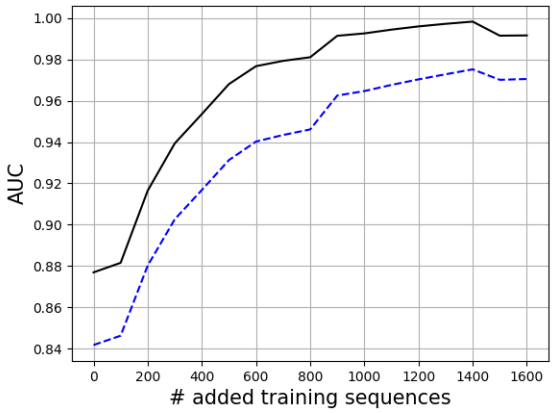


Fig. 7. ADFA-LD dataset: AUC curves as a function of the number of normal data added to the initial training set. The black continuous line corresponds to the situation for which the sequences of attacks that are an exact subsequence of a normal data have been removed. The blue dotted line corresponds to the case where no sequence of attacks has been removed.

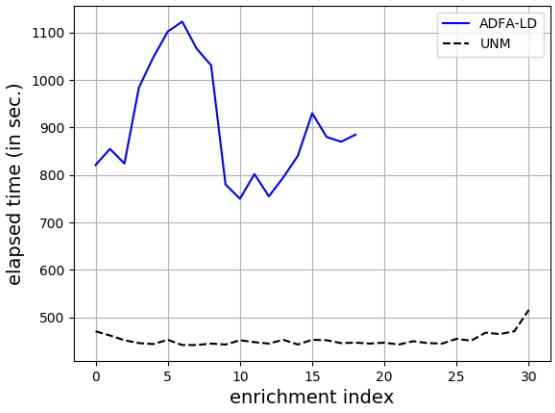


Fig. 8. Elapsed time in seconds during each step of the enrichment procedure for the UNM (dotted black line) and ADFA-LD (continuous blue line) datasets.

sequences using the 1000 sequences of the remaining normal data having the lowest covering similarity. From top to bottom, we show that the model improves its capacity to separate attack data from normal data: the AUC value that is initially $.84/.88$, reaches $.93/.97$ when 500 normal sequences have been added to the initial training data and $.96/99$ when 1000 normal sequences have been added to the 833 initial training set. Similarly as observed for the UMN dataset, in the middle column, we see that the similarity score for the normal data is progressively tangent to the 1 constant curve, while, for the attack data, it generally stays much lower, although it does increase slightly.

Fig. 7 presents for this experiment the AUC values for the algorithm as the enrichment of the training data increases. We can see on this figure that the SC4ID algorithm improves rapidly until reaching an almost perfect separation (when the attacks that are subsequences of the training data have been removed) of the normal and attack data when 1400 normal data have been used to enrich the initial 833 training set. Here again, the instance selection that is performed directly from the covering similarity scores is working particularly well. Nevertheless, we notice after adding 1400 sequences that the algorithm is a little less efficient with an AUC value that drops from $.975/.998$ to $.971/.992$. The explanation is that a remaining few attacks (precisely the 11 sequences mentioned earlier that have a covering of size 2 when using only the 833 training sequences) become subsequences of some of the last added normal subsequences to the training set. The similarity for these few attacks reaches the maximal value, namely 1.0, and they cannot be separated anymore, hence a lower AUC value.

4.3 Setting up the σ parameter

The covering similarity being normalized in $[0, 1]$, the choice for σ is constrained in the unit interval. Fig. 2 and Fig. 6 show that, when enough normal training data has been selected using the procedure that consists of selecting first the normal sequences having the lowest similarity score, we can reach an almost perfect separation between attack and normal sequences using $\sigma \lessapprox 1.0$. According to our experiments on system calls, selecting $\sigma = .97$ will lead to the result that small size coverings (no more than 3% of the size of the test sequence) will lead to a high detection rate with a very low false alarm rate. It also leads to a relatively small training set since only about 30% of the available normal data are used for training in this case. We doubt that a general rule for selecting σ exists. The fine-tuning of σ is application-dependent.

4.4 Runtime consideration

We have not been able to provide a comparative study on the ADFA-LD dataset, due to the too high time complexity of the three other similarities (LEV, LCSt, LCSq). In particular, given the dataset statistics presented in Table 1 and the quadratic complexity of LEV and LCSq measures, we estimate that, if we use these two measures, it would require about 5 months of computation for achieving the results on a simple core of Intel i7 architecture, while it takes about 15 minutes for SC4ID using the same hardware. Consequently, we only discuss hereafter runtime matters for the SC4ID algorithm.

Fig. 8 presents the elapsed time expressed in seconds for each iteration of the enrichment process carried out by SC4ID for the UNM (dotted curve) and ADFA-LD (continuous curve) datasets. Each step of this process involves the construction of a suffix-tree based on the training (normal) data, S, and the extraction of the S-optimal coverings for the validation (normal) and attack data. These curves have been obtained on an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz laptop running an Ubuntu 16.10 release. Note that between two successive tests, the training data increases by one sequence for the UNM dataset and 100 sequences for the ADFA-LD dataset; conversely the validation (normal) dataset is reduced by one sequence and 100 sequences respectively. Hence, as the enrichment increases, the number of S-optimal coverings to extract decreases.

We observe that the elapsed time is quite stable, around 450 seconds, for the UNM dataset and is rather independent of the enrichment step. For the ADFA-LD dataset (which is much larger in size than the UNM dataset), the elapsed time varies around an average of about 900 seconds with a large standard deviation (about 100 sec.). From the analysis of the logs collected during our experiments, we found that i) the time required to construct the suffix-tree is marginal, ii) the elapsed time progressively shifts from the extraction of the coverings for the normal sequences of the validation set to the extraction of the coverings for the attack data. This is due partly to the fact that the amount of normal validation data to evaluate decreases, but also because the extraction is becoming much easier for the normal data because the size of the covering becomes very small in average as the enrichment progresses (similarity scores are tangent to the maximal similarity score for normal data).

Due to the variability in length of the sequences that are added to the training data, it is difficult to progress deeper into this analysis. Nevertheless, these curves show that, i) for the various configurations corresponding to each step of the enrichment process, the extraction of the S-optimal coverings is bounded in time and ii) the two addressed problems are processed in a quite reasonable elapsed time.

5 Discussion

The UNM dataset, although relatively old and somewhat outdated, gives a general view of the ability of the SC4ID algorithm to separate normal data from attacks sequences. Using this this dataset, we have been able to compare SC4ID with three other similarity measures whose variants are classically used in bioinformatics and text mining. From this comparative study we can conclude that SC4ID is as accurate as the Levenshtein's distance but is much faster to evaluate. SC4ID is also much more accurate and faster to evaluate than the two other similarity measures.

We have not been able to provide a similar comparative study on the ADFA-LD dataset due to the time complexity of the other similarity measures.

The fact that the first results highlighted on the UNM dataset for the SC4ID algorithm are still observable on a much newer and reputedly difficult benchmark such as the ADFA-LD dataset is particularly promising. In both cases, the instance selection ability of SC4ID enables the rapid improvement of the separability of attack sequences from the normal ones. When the training data are sufficiently representative of the normal activity, expressed in terms of system call traces, the algorithm performs an almost perfect separation. We have shown that, if the perfect separation between normal and attack sequences is not achieved for the ADFA-LD dataset, this is because some attack sequences actually correspond exactly to some subsequences of the normal sequences that are used to train the model. If this kind of overlap is suppressed then one can expect a perfect separability as shown in Fig. 7.

In addition, the covering similarity can be used efficiently to decide whether a given normal sequence should be part of the training data or not. Basically, when a lot of normal data is available, which is actually the case for HIDS application, it offers a data selection scheme with a stop condition that can be easily set up according to the middle columns of Fig. 2 and 6. When the ranked normal similarity curve is tangent to the 1-constant horizontal line, SC4ID will not further improve its capability to isolate anomaly data (anomaly being clearly defined in the context of the considered set of normal data).

To give some hints about how SC4ID compares with the state of the art methods that have been tested on the ADFA-LD dataset, we report hereinafter the results we found in the recent literature:

In [46], bag of words and vector space models with *tf* and *tf-idf* weightings were used. Authors report a slightly less than 80% in accuracy for a False Positive rate of 30%.

In [47], a one-class SVM was evaluated using a feature vector composed of n-grams of length 5. The authors report an average accuracy of 70% at a False Positive Rate of about 20%.

In [48], a 10 fold cross-validation supervised classification (which is a much easier task than the anomaly detection task we are considering in this study, since attack data is used to train the classifiers) has been conducted. Authors report a AUC=0.93 for the best tested method (k-nn with k=3), using an 'enhanced' vector space model with n-grams (n=2,3,4,5).

In [31], a deep learning ensemble approach (LSTM) has been evaluated. Authors report an AUC value of 0.928 for an aggregating method that is somewhat questionable, and an AUC value of 0.859 for a classical voting ensemble method. This implies that for a single LSTM model, the AUC value would be at most $.86$.

In [34], a combinatorics 'semantic' approach has been set up. It requires enumerating all phrases of 5 words with gaps, each word consisting of any subsequence (of any length) extracted from the training data (we estimate at about 6.10^16 the number of such phrases). The authors, actually the designers of the ADFA-LD dataset, report an $AUC = .95$ after several weeks of computing effort.

In [49], authors proposed to reduce system call traces by applying trace abstraction techniques. The approach consists mainly of reducing the size of the alphabet, by using meta-symbols, i.e. subsets that partition the alphabet. Authors report a 12.69% False Positive rate for a 100% True Positive rate when using a HMM model.

All these results demonstrate that SC4ID is well positioned among leading-edge approaches.

6 Conclusion

In this paper, we have presented the design and implementation of SC4ID, a novel algorithm based on the concept of sequence covering, to detect abnormal sequences of system calls. It relates to a semi-supervised learning approach that it is quite efficient to detect zero-day attacks, as far as enough normal training data is available. Its main advantages compared to the state of the art approaches are:

i) it is parameter-free, except for the decision threshold σ. Hence no assumption need to be made on the data, no windowing, no maximal length for the n-grams that are taken into account, no hidden architecture (HMM,

LSTM) need to be defined, no meta parameters (SVM, RF) need to be tuned, etc.,

ii) the S-optimal covering provided by the measure is interpretable and can be used to locate contextual and collective anomalies in long sequences,

iii) it is incremental: the elements of the covering are progressively discovered and never modified afterwards. Hence the algorithm can be easily setup for an online exploitation,

iv) it enables a fast learning curve, using the efficient sequence selection procedure that can be used to sample the (large)set of normal sequences in order to build a minimal training set,

v) it scales well and runs easily on common hardware for medium to large size problems, thanks to its $n \cdot log(n)$ algorithmic complexity.

However, we have pointed out a limitation of the SC4ID algorithm: it cannot separate sequences that are exact subsequences of the training set. However, any algorithm that uses historical data to provide a regressive or predictive scoring would also fail to correctly handle such a situation.

As a perspective, we can expect a speed up of the algorithm when trace abstraction techniques, such as described in [49], are used to reduce system call traces. This speed-up, hopefully, could potentially be obtained without losing (too much) on the accuracy. Parallelization of the algorithm is also an issue that can be addressed, in particular in the context of suffix-array implementations.

Finally, one may ask whether the covering similarity is able to offer some useful solutions in other domain such as bioinformatics, sequence mining (clustering, classification), plagiarism detection, etc. In particular, the pair-wise similarity defined in Eq. 2 could be tested in various context, to evaluate the robustness and generality aspect of the concept of sequence covering.

REFERENCES

[1] A. Regev and T. Darsan, "Tinba malware reloaded and attacking banks around the world," *SecurityIntelligence*, 2014.

[2] D. Bonderud, "Tinba number five takes aim at asia-pacific finance," *SecurityIntelligence*, 2016.

[3] ——, "Malware top 10: Conficker grabs top spot, tinba takes second," *SecurityIntelligence*, 2016.

[4] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, March 1981. [Online]. Available: http://gel.ym.edu.tw/~chc/AB_papers/03.pdf

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, Mar. 1970. [Online]. Available: http://dx.doi.org/10.1016/0022-2836(70)90057-4

[6] I. Korf, M. Yandell, and J. Bedell, *BLAST*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2003.

[7] E. Lamperti, J. Kittelberger, T. Smith, and V.-K. L, "Corruption of genomic databases with anomalous sequence," *Nucleic Acids Research*, vol. 20, no. 11, pp. 2741–2747, 1992. [Online]. Available: https://doi.org/10.1371/journal.pone.0186251

[8] Q. Quan, W. Jinlin, Z. Wei, and X. Mingjun, "Improved edit distance method for system call anomaly detection," in *2012 IEEE 12th International Conference on Computer and Information Technology*, Oct 2012, pp. 1097–1102.

[9] T. Ho, S.-R. Oh, and H. Kim, "A parallel approximate string matching under levenshtein distance on graphics processing units using warp-shuffle operations," *PLOS ONE*, vol. 12, no. 10, pp. 1–15, 10 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0186251

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[11] K. A. Scarfone and P. M. Mell, "Sp 800-94. guide to intrusion detection and prevention systems (idps)," Gaithersburg, MD, United States, Tech. Rep., 2007.

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

[13] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Review: Intrusion detection system: A comprehensive review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 16–24, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2012.09.004

[14] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, First 2014.

[15] E. Hodo, X. J. A. Bellekens, A. Hamilton, C. Tachtatzis, and R. C. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," *CoRR*, vol. abs/1701.02145, 2017. [Online]. Available: http://arxiv.org/abs/1701.02145

[16] 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, Aug. 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298081.1298084

[17] Y. Wang, J. Wong, and A. Miner, "Anomaly intrusion detection using one class svm," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, June 2004, pp. 358–364.

[18] Y. Yao, Y. Wei, F. x. Gao, and G. Yu, "Anomaly intrusion detection approach using hybrid mlp/cnn neural network," in *Sixth International Conference on Intelligent Systems Design and Applications*, vol. 2, Oct 2006, pp. 1095–1102.

[19] S. Budalakoti, A. N. Srivastava, and M. E. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 101–113, Jan 2009.

[20] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, feb 1966, doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[21] S. Coull, J. Branch, B. Szymanski, and E. Breimer, "Intrusion detection: A bioinformatics approach," in *Proceedings of the 19th Annual Computer Security Applications Conference*, ser. ACSAC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 24–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956415.956442

[22] A. Kundu, S. Sural, and A. K. Majumdar, "Database intrusion detection using sequence alignment," *International Journal of Information Security*, vol. 9, no. 3, pp. 179–191, Jun 2010. [Online]. Available: https://doi.org/10.1007/s10207-010-0102-5

[23] M. D. Wan, S. H. S. Huang, and J. Yang, "Finding the longest similar subsequence of thumbprints for intrusion detection," in *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, vol. 1, April 2006, pp. 6 pp.–.

[24] A. Sureka, "Kernel based sequential data anomaly detection in business process event logs," *CoRR*, vol. abs/1507.01168, 2015. [Online]. Available: http://arxiv.org/abs/1507.01168

[25] Y. Qiao, X. W. Xin, Y. Bin, and S. Ge, "Anomaly intrusion detection method based on hmm," *Electronics Letters*, vol. 38, no. 13, pp. 663–664, Jun 2002.

[26] J. Hu, *Host-Based Anomaly Intrusion Detection*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 235–255. [Online]. Available: https://doi.org/10.1007/978-3-642-04117-4_13

[27] R. Jain and N. S. Abouzakhar, "Hidden markov model based anomaly intrusion detection," in *2012 International Conference for Internet Technology and Secured Transactions*, Dec 2012, pp. 528–533.

[28] K. K. Gupta, B. Nath, and K. Ramamohanarao, "Conditional random fields for intrusion detection," in *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, vol. 1, May 2007, pp. 203–208.

[29] L. Taub, "Applying conditional random fields to payload anomaly detection with crfpad," in *2013 Proceedings of IEEE Southeastcon*, April 2013, pp. 1–5.

[30] M. Sheikhan, Z. Jadidi, and A. Farrokhi, "Intrusion detection using reduced-size rnn based on feature grouping," *Neural*

Computing and Applications, vol. 21, no. 6, pp. 1185–1190, Sep 2012. [Online]. Available: https://doi.org/10.1007/s00521-010-0487-0

[31] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*, Feb 2016, pp. 1–5.

[32] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, Feb 2007. [Online]. Available: https://doi.org/10.1007/s11416-006-0030-0

[33] B. Orisaniya and D. Patel, "Evaluation of modified vector space representation using adfa-ld and adfa-wd datasets." *Journal of Information Security*, vol. 6, pp. 250–264, 2015.

[34] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, April 2014.

[35] ——, "Generation of a new ids test dataset: Time to retire the kdd collection," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 4487–4492.

[36] P. Bieganski, J. Riedl, J. V. Cartis, and E. F. Retzel, "Generalized suffix trees for biological sequence data: applications and implementation," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 5, Jan 1994, pp. 35–44.

[37] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, Sep 1995. [Online]. Available: https://doi.org/10.1007/BF01206331

[38] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '90. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990, pp. 319–327. [Online]. Available: http://dl.acm.org/citation.cfm?id=320176.320218

[39] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays," *Journal of Discrete Algorithms*, vol. 2, no. 1, pp. 53 – 86, 2004, the 9th International Symposium on String Processing and Information Retrieval. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866703000650

[40] S. Forrest, "Intrusion detection dataset. university of new mexico (unm)," 1998. [Online]. Available: http://www.cs.unm.edu/~immsec/systemcalls.htm

[41] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974. [Online]. Available: http://doi.acm.org/10.1145/321796.321811

[42] D. Maier, "The complexity of some problems on subsequences and supersequences," *J. ACM*, vol. 25, no. 2, pp. 322–336, Apr. 1978. [Online]. Available: http://doi.acm.org/10.1145/322063.322075

[43] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*, 2000, pp. 39–48.

[44] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. New York, NY, USA: Cambridge University Press, 1997.

[45] E. N. Yolacan, "Learning from sequential data for anomaly detection," Ph.D. dissertation, Boston (Mass.) : Northeastern University, 2014.

[46] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of adfa-ld," in *2013 6th International Congress on Image and Signal Processing (CISP)*, vol. 03, Dec 2013, pp. 1711–1716.

[47] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld," in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Aug 2014, pp. 978–982.

[48] B. Borisaniya and D. Patel, "Evaluation of modified vector space representation using adfa-ld and adfa-wd datasets," *Journal of Information Security*, vol. 6, pp. 250–264, 2015.

[49] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and S. Gagnon, "A trace abstraction approach for host-based anomaly detection," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, May 2015, pp. 1–8.