MORPHEUS: **M**otif **O**riented **R**epresentations to **P**urge **H**ostile **E**vents from **U**nlabeled **S**equences

Gaurav Tandon Philip Chan Debasis Mitra

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
USA

{gtandon, pkc, dmitra}@cs.fit.edu

ABSTRACT

Most of the prevalent anomaly detection systems use some training data to build models. These models are then utilized to capture any deviations resulting from possible intrusions. The efficacy of such systems is highly dependent upon a training data set free of attacks. “Clean” or labeled training data is hard to obtain. This paper addresses the very practical issue of refinement of unlabeled data to obtain a clean data set which can then train an online anomaly detection system.

Our system, called MORPHEUS, represents a system call sequence using the spatial positions of motifs (subsequences) within the sequence. We also introduce a novel representation called sequence space to denote all sequences with respect to a reference sequence. Experiments on well known data sets indicate that our sequence space can be effectively used to purge anomalies from unlabeled sequences. Although an unsupervised anomaly detection system in itself, our technique is used for data purification. A “clean” training set thus obtained improves the performance of existing online host-based anomaly detection systems by increasing the number of attack detections.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representations (procedural and rule-based)

I.2.6 [**Learning**]: Parameter Learning

General Terms

Algorithms, Security.

Keywords

Anomaly detection, motifs, data cleaning.

1. INTRODUCTION

Anomaly detection, models normal behavior of applications and significant deviations from this behavior are considered anomalous. Anomaly detection systems can detect novel attacks but also generate false alarms since not all anomalies are hostile. Monitoring system call sequences has been successful in detecting process-based anomalies corresponding to attacks. But all the proposed techniques rely on “clean” training data to build their model. Current audit sequence is then examined for anomalous behavior. An attack embedded inside the training data would result in an erroneous model, since all future occurrences of the attack would be treated as normal. Purging all malicious content from audit data is hence imperative.

There are two distinct tasks in any anomaly detection system: (i) Cleaning data for training -This is the preprocessing step and can be performed offline. (ii) Detecting novel attacks - This is an online process where an alarm is raised when an anomaly is detected by the system. Offline cleaning can tolerate higher false alarm rate than online detection because the goal of cleaning is to obtain an attack-free training data set (though false alarms can reduce the size of the training set). Online detection of novel attacks desires low false alarm rate, otherwise the administrator might ignore the detector.

Unsupervised learning techniques have been proposed in the field of network anomaly detection but have not been well researched in host-based systems. In this paper, we present a representation for a system call sequence using the spatial relationships between the motifs (subsequences) occurring in the sequence. Utilizing this representation, we propose a novel way to represent system call sequences (called sequence space), which can eventually be used to determine anomalies. Our system, called MORPHEUS (**M**otif **O**riented **R**epresentations to **P**urge **H**ostile **E**vents from **U**nlabeled **S**equences), is an anomaly detector itself. It refines data to train other anomaly detection systems.

Our contributions are as follows:

(i) Motif representation - We represent sequences on the basis of the motifs and introduce the concept of a sequence space. Empirical results indicate that our representation can be effectively used to detect malicious sequences from the data using unsupervised learning techniques.

(ii) Parameter for LOF (details in Section 3.5) - We propose heuristics to automate the process of determining the parameters to the existing outlier detection algorithm.

(iii) We show empirically that cleaning training data improves performance of existing online detection systems.

VizSEC/DMSEC'04, October 29, 2004, Washington DC, USA.

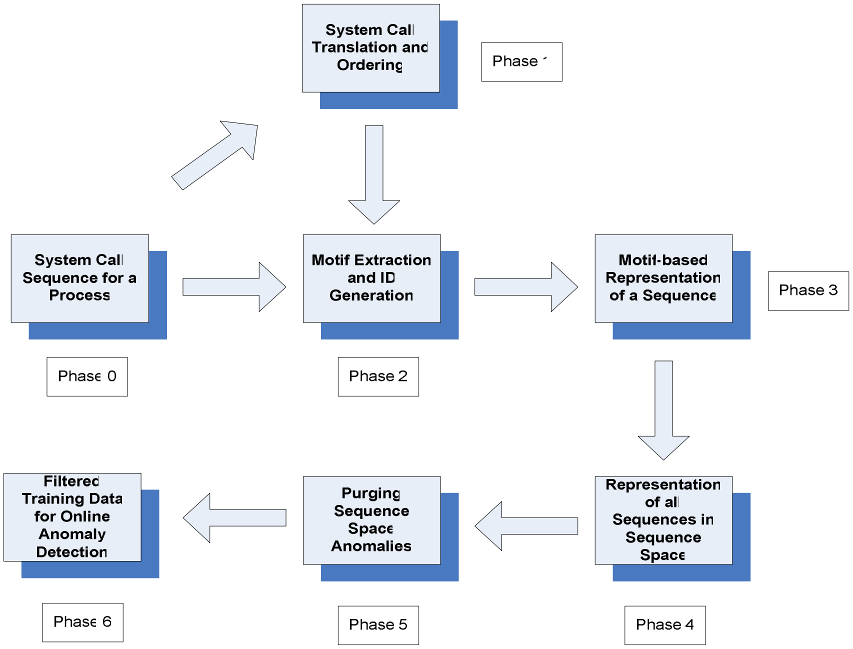

System Call Translation and Ordering
Phase 1
System Call Sequence for a Process
Phase 0
Motif Extraction and ID Generation
Phase 2
Motif-based Representation of a Sequence
Phase 3
Representation of all Sequences in Sequence Space
Phase 4
Purging Sequence Space Anomalies
Phase 5
Filtered Training Data for Online Anomaly Detection
Phase 6

Figure 1. System Architecture of MORPHEUS.

The paper is organized as follows. In Section 2, we review some anomaly detection systems. In Section 3, we present the system architecture and detail the various phases of MORPHEUS. In Section 4, we summarize and analyze the results obtained from the experiments performed on synthetic and real data sets. In Section 5, we conclude and put forth some issues we plan to address in the future.

2. RELATED WORK

Traditional host based anomaly detection techniques create models of normal behavioral patterns and then look for deviations in test data. Such techniques perform supervised learning. Forrest et al. [10] memorized normal system call sequences using a look-ahead pairs. Lane and Brodley [17, 18] examined UNIX command sequences to capture normal user profiles using a fixed size window. Later work by Warrender et al. [35] extended sequence modeling by using n-grams and their frequency. Wespi et al. [36, 37] proposed a scheme with variable length patterns using Teiresias [29], a pattern discovery algorithm in biological sequences. Ghosh and Schwartzbard [11] used artificial neural networks, Sekar et al. [31] proposed a finite state automaton, Jiang et al. [13] also proposed variable length patterns, Liao and Vemuri [21] used text categorization techniques, Jones and Li [14] learnt temporal signatures, Coull et al. [7] suggested sequence alignment, Mazeroff et al. [25] proposed probabilistic suffix trees, and Lee at al [20] used machine learning algorithm called RIPPER [6] to learn normal user behavior. All these techniques require "clean" or labeled training data to build models of normal behavior, which is hard to obtain. The data sets used are synthetic and generated in constrained environments. They are not representative of actual application behavior, which contains many irregularities. The need for a system to filter audit data and produce a "clean" data set motivates our current research.

Unsupervised learning is an extensively researched topic in network anomaly detection [27, 9, 5, and 19]. Network traffic comprises of continuous and discrete attributes which can be considered along different dimensions of a feature space. Distance and density based algorithms can then be applied on this feature space to detect outliers. Due to the lack of a similar feature space, not much work has been done using unsupervised learning techniques in host based systems. In this paper, we present a novel framework of system call sequences (called sequence space) and demonstrate the efficacy of our representation to purge outliers using unsupervised learning techniques.

3. SYSTEM ARCHITECTURE

In this section, we present the system architecture and describe the various stages involved in our system. The overview of our system is presented in Figure 1.

Audit sequences corresponding to all the processes for an application are the inputs to our system (Phase 0). Every sequence is a process in execution and initial preprocessing might be required to obtain them. The preprocessing of audit data is explained in Section 4.1. In the first phase, all the unique system calls are extracted and mapped to a unique id. Using the sequences and the system call mapping table, we extract motifs (subsequences) which are either repetitive within the same sequence or are common across any two sequences (Phase 2). Once all motifs have been extracted and inserted into a database, they are ranked and assigned a unique id.

Using the motif database, we create a representation for a sequence by recording all the motifs that occur within that sequence and their corresponding positions (Phase 3). In Phase 4, these representations are used to map all sequences in a single plot (called *sequence space*). Anomaly detection is performed on the sequence space and outliers are detected (Phase 5). These suspicious components are removed to obtain a relatively "clean" data set, which is then fed as training input to an online detection algorithm in Phase 6.

3.1 Phase 1: Translation and ordering of system calls

We represent a process (system call sequence) as a finite sequence of system calls, where each system call belongs to the finite set ∑. A system call sequence (SCS) *s* is thus represented as (c_1 c_2 c_3 . . . c_n), where $\mathrm{c}_{\mathrm{i}} \in \Sigma, 1 \leq i \leq n.$ |*s*| represents the length of SCS *s*.

As a result of the pre-processing stage, we obtain system call sequences as finite (but not necessarily equal) length strings. We then map each system call to a unique symbol using a translation table. Once we have mapped all the system calls, we rank their corresponding ids by utilizing prior knowledge as to how susceptible the system call is to malicious usage. Bernaschi et al. [3] proposed kernel enhancements to Linux using a threat level classification. We use a similar ranking scheme which allows system calls with similar threat levels to be grouped together. This ranking is used in Phase 2.

3.2 Phase 2: Motif extraction and id generation

A motif is defined as a subsequence of length greater than *p* if it appears more than *k* times, for some positive integers *p* and *k*, within the finite set $S = \{s_1, s_2, . . . s_m\}$ comprising *m* SCSs. Now our task is reduced to extract motifs (subsequences) from the strings (corresponding to the system call sequences). We extract two sets of motifs via "auto-match" and "cross-match".

3.2.1 Motif extraction using auto-match

Our first set of motifs comprises the frequently occurring patterns within each sequence. For our experiments, we considered any pattern at least 2 characters long, occurring more than once as frequent. While the set of SCSs *S* is the input to this algorithm, a set of unique motifs $M=\{m_1, m_2, . . ., m_q\}$ is the output. It may happen that a smaller length subsequence is subsumed by a longer one. We prune the smaller motif only if it is not more frequent than a larger motif that subsumes it.

Definition A motif m_i extracted using auto-match (1) has length ≥ 2, (2) has frequency ≥ 2, and (3) if there exists a motif $m_j \in M$ in a sequence $s_k \in S$ such that m_i is a subsequence of m_j but occurs independently in SCS s_k.

To illustrate this idea, consider the following synthetic sequence

acggcggfgjcggfgjxyz (I)

The frequently occurring subsequences (with their respective frequency) are *cg*(3), *gg*(3), *gf*(2), *fg*(2), *gj*(2), *cgg*(3), *cggf*(2), *ggfg*(2), *gfgj*(2), *cggfg*(2), *ggfgj*(2), *cggfgj*(2). The longest pattern *cggfgj* subsumes all the smaller subsequences except *cg*, *gg* and *cgg* since they are more frequent than the longer pattern, implying independent occurrence. But *cg* and *gg* are subsumed by *cgg*, since they all have the same frequency. Thus, the final set of motifs *M*={*cgg*, *cggfgj*}.

We start by looking at such sub-strings of length two within each string. Then, we do the same for substrings of increasing lengths. Two or more overlapping motifs may be merged together to form a motif of greater length. This is how we create motifs of arbitrarily lengths. Representing sequences with possibly overlapping motifs is based on Allen's temporal reasoning scheme [2]. After extracting motifs of length 4 in the example sequence (I), we have motifs *cggf*, *ggfg* and *gfgj*, all with frequency 2. Since these patterns are overlapping and have the same frequency, they may be merged together in order to obtain a longer motif *cggfgj* with the same frequency 2. However, there are instances when the smaller motifs may concatenate forming a longer motif at some places, but may occur at other positions on the sequences independently (i.e., not overlapping). Even though they have the same frequency, the concatenated longer motif does not subsume the shorter ones. Consider the sequence c*ggfgjabcggfpqrggfgxyzgfgj*. Here, the motifs *cggf*, *ggfg* and *gfgj* each have a frequency 2. But the longer motif *cggfgj* occurs only once, though we may wrongly conclude a frequency of 2 by using the above derivation. The remaining instances of the smaller motifs are at different (non-overlapping) positions within the string. The solution to this problem is that the occurrence of the longer motif obtained from the fusion of the smaller motifs should be verified for accuracy. If the frequency of the longer motif is found to be the same as that of a smaller one, then merging of the latter ones is all right, i.e., we ignore the smaller motifs. This technique reduces the effort of going through all possible string lengths and of finding all possible motifs for those lengths. The procedure of finding motifs of variable lengths by first merging and then verifying is repeated until no more motifs could be merged.

3.2.2 Motif extraction using cross-match

Apart from frequently occurring patterns, we are also interested in patterns which do not occur frequently but are present in more than one SCS. We believe that these motifs could also be instrumental in modeling an intrusion detection system since they reflect common behavioral patterns across sequences. We performed pair-wise cross-match between different sequences to obtain these.

Definition A motif m_i extracted using cross-match (1) has length ≥ 2, (2) appears in at least a pair of sequences $s_k, s_l \in S$, and (3) is maximal, i.e., there does not exist a motif $m_j \in M (j \neq i)$ such that $m_j \subseteq s_k, s_l$ and $m_i \subset m_j$.

Let us consider the following pair of synthetic sequences:

acgfgjcgfgjxyzcg (II)

cgfgjpqrxyzpqr (III)

Using cross-match between the example sequences (II) and (III), we get the motifs *cgfgj* and *xyz*, since these are the maximal common subsequences across the two given sequences.

A simple method for comparing amino acid and nucleotide sequences called the Matrix Method is described by Gibbs and McIntyre [12]. A matrix is formed with one sequence written across and the other in the downward position on the left of the matrix. Any common element was marked with a dot and a series of dots along a diagonal gave a common subsequence

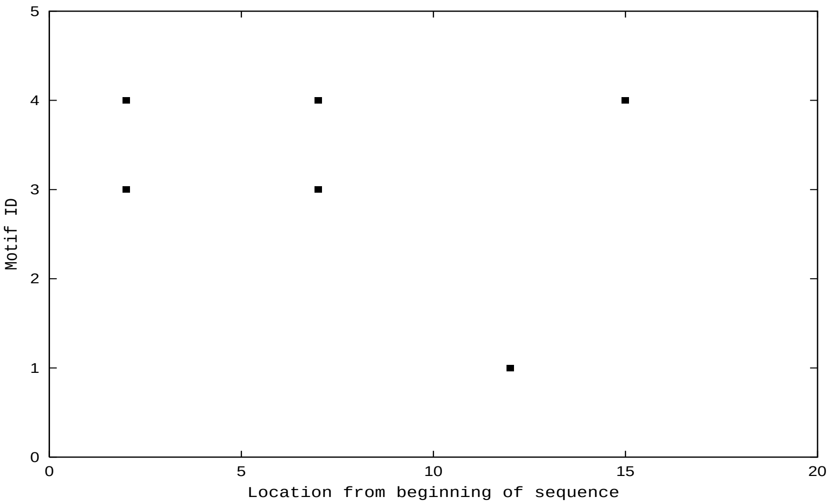


Figure 2. Motif-oriented representation for sequence (II).

between the two sequences. Using a technique similar to the Matrix Method, we extract motifs which occur across sequences but may not be frequent within a single sequence itself.

After generating all the motifs for a sequence (auto-match) or pairs of sequences (cross-match), we added them to the motif database and pruned redundant motifs. We ordered the motifs on the likelihood of involving in an attack using a dictionary sort and the ranking of the system calls in Section 3.1. The motifs are then assigned a unique id based upon their position within the ordered motif database.

3.3 Phase 3: Motif-based representation of a sequence

Once we have all the motifs that exist in the set *S* of sequences in the motif database *M*, we would like to represent each sequence in terms of the motifs existing within it. For each sequence $s_i \in S$, we list all the motifs occurring within it along with their starting positions within the sequence.

This creates a two dimensional representation for each SCS s_i, where the X-axis is the distance along the sequence from its beginning, and the Y-axis is the motif id of those motifs present in s_i. With this scheme a sequence could be visualized as a scatter plot of the motifs present in a sequence. Figure 2 depicts such a representation for the synthetic sequence (II), where the motifs *cg*, *cgfgj* and *xyz* are represented at the positions of occurrence within the respective sequence. A motif's starting point is the abscissa and the motif ID is the ordinate of the corresponding point in the scatter plot. A total of 4 unique motifs (*cg*, *cgfgj*, *pqr* and *xyz*), obtained from auto-match and cross-match of (II) and (III), are assumed in the motif database for the plot in Figure 2. This representation is for visualization purposes only. At the end of this phase, our system stores each SCS as a list of all motifs present within along with their spatial positions from the beginning of the sequence.

Once we have all the motifs that exist in the set *S* of sequences in the motif database *M*, we would like to represent each sequence in terms of the motifs existing within it. For each sequence $s_i \in S$, we list all the motifs occurring within it alongwith their starting positions within the sequence.

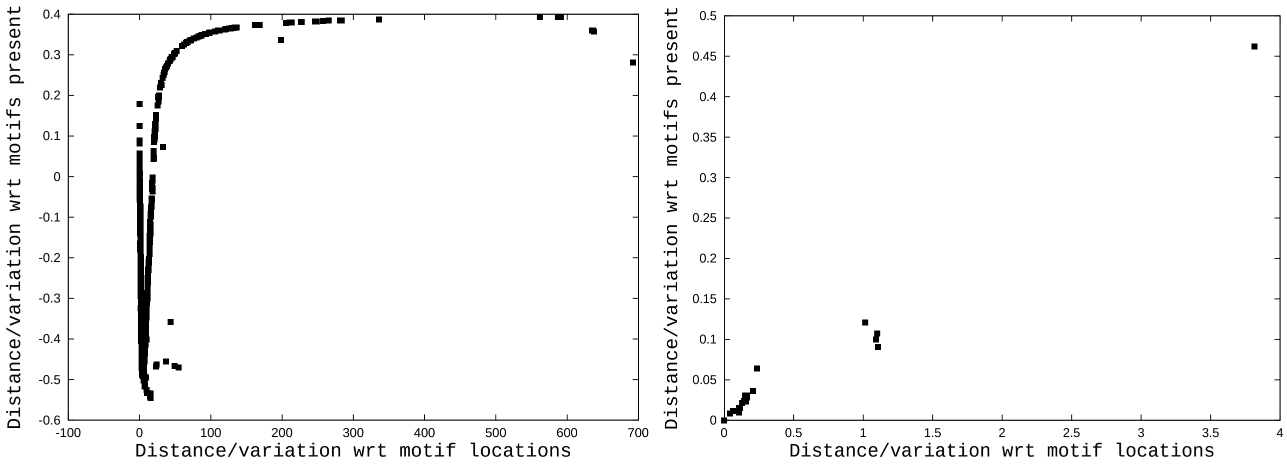
3.4 Phase 4: Sequence space – a single representation for all sequences

We model all sequences on the basis of their motifs. Malicious activity results in alterations in the SCS which is reflected by the variations in the motifs and their spatial positions. Plotting all the SCSs (based upon their motif representations) in a single feature space could reflect the similarity/dissimilarity between them.

After creating a motif-based representation for each sequence (Section 3.3), we plot all the test sequences *S* in a feature space called the *sequence space*. In this representation we measure the distance between pairs of SCSs along each of the two axes (motifs and their locations). Utilizing one (arbitrarily chosen) SCS from the set *S* as a reference sequence s_1, we measure (d_x, d_y) distances for all SCSs $s_i \in S$. Thus, the sequences are represented as points in this 2D sequence space, where the sequence s_1 is at the origin (reference point) on this plot. Let s_2 be any other sequence in *S* whose relative position with respect to s_1 is to be computed. Inspired by the symmetric Mahalanobis distance [23], the distance is computed as follows:

$$d_x = \frac{\dfrac{\sum_{i=1}^{n_1}(x_{1_i} - \bar{x}_2)}{\sigma_{x_2}} + \dfrac{\sum_{j=1}^{n_2}(x_{2_j} - \bar{x}_1)}{\sigma_{x_1}}}{n_1 + n_2}$$

$$d_y = \frac{\dfrac{\sum_{i=1}^{n_1}(y_{1_i} - \bar{y}_2)}{\sigma_{y_2}} + \dfrac{\sum_{j=1}^{n_2}(y_{2_j} - \bar{y}_1)}{\sigma_{y_1}}}{n_1 + n_2} \qquad \text{(IV)}$$



(a) **(b)**

Figure 3. Sequence space for (a) ftpd, and (b) lpr applications.

where s_1 has n_1 motif occurrences and s_2 has n_2 motif occurrences, (d_x, d_y) is the position of s_2 w.r.t. s_1, $(\bar{x}, \bar{y})$ is the mean and (σ_x, σ_y) is the standard deviation along the x and y axes. Using this metric, we try to calculate the variation in motifs and their locations in the two sequences.

After computing (d_x, d_y) for all sequences in S with respect to the reference sequence (s_1), we plot them in the sequence space, as represented by the two plots in Figure 3. The origin represents the reference sequence It is important to note that the position of another sequence (calculated using IV) with respect to the randomly selected reference sequence can be negative (in X and/or Y direction). In that case the sequence space will get extended to other quadrants as well, as in Figure 3(b).

3.5 Phase 5: Purging sequence space anomalies

Similar sequences are expected to cluster together in the sequence space. Malicious activity is known to produce irregular sequence of events. These anomalies would correspond to spurious points (global outliers) or local outliers in the scatter plot created in Phase 4. In Figure 3(a), the point on the top-right corner of the plot is isolated from the rest of the points, making it anomalous. In this section we will concentrate on outlier detection, which has been well researched topic in databases and knowledge discovery [16, 4, 28, and 1].

LOF [4] is a density-based outlier finding algorithm which defines a local neighborhood, using which a degree of outlierness is assigned to every object. A reachability distance is computed for every object based upon its distance from its k^{th}-nearest neighbor. A reachability density is then calculated for every object based upon the average reachability distance of that object from its neighbors (number of neighbors – MinPts –being an input parameter). Finally, a local outlier factor (LOF) is associated with every object by comparing its reachability density with each of its neighbors. A local outlier is one whose neighbors have a high reachability density as compared to that object. For each point this algorithm gives a degree to which that point is an outlier as compared to its neighbors. This LOF score corresponds to the anomaly score of that point in our model. Our system computes the anomaly scores for all the SCSs (represented as points in sequence space). All the points for which the score is greater than a threshold are considered anomalous and removed.

3.5.1 Automating the LOF parameters

3.5.1.1 MinPts

We use LOF for anomaly detection in the sequence space. LOF takes MinPts as an input parameter, which signifies the number of neighborhood points to be compared with. The performance of the system is sensitive to the parameter MinPts. A human expert (in our case a system administrator) would be required to analyze the sequence space and then come up with a reasonable value of MinPts. But the LOF values increases and decreases non-monotonically [4]. So it is highly desirable for this parameter selection to be automated. We present one intuitive way in which this can be computed without the help of any human expert. To select MinPts, we use clustering to identify the larger neighborhoods. Then, we scrutinize each cluster and approximate the number of neighbors in an average neighborhood.

(a) Finding the number of clusters: After creating the sequence space, we use the L-Method [30] to predict the number of clusters in the representation. This is done by creating a "number of clusters vs. merge distance" graph obtained from merging one data point at a time in the sequence space. Starting with all N points in the sequence space, the 2 closest points are merged to form a cluster. At each step, a data point with minimum distance to another cluster or data point is merged. At the final step, all points are merged into the same cluster. The graph obtained has 3 distinct areas – a horizontal region (points/clusters close to each other merged), a vertical region (far away points/clusters merged), and a curved region in between. The number of clusters is represented by the knee of this curve, which is the intersection of a pair of lines fitted across the points in the graph that minimizes the root mean square error. Further details can be obtained from [30].

(b) Calculating MinPts: Assume k clusters are obtained in a given sequence space using L-Method (with each cluster containing at least 2 points). Let α_i be the actual number of points in cluster $i, 1 \le i \le k$. Let ρ_i be the maximum pair-wise distance between any 2 points in cluster i; and τ_i is the average (pair-wise) distances between 2 points in cluster i. Let β_i be the expected number of points in cluster i. Its value can be computed by dividing the area of the bounding box for the cluster with the average area occupied by the bounding box of any 2 points in the cluster (for simplicity we assume square shaped clusters). Therefore, we get

$$\beta_i = \left(\frac{\rho_i}{\tau_i}\right)^2 \qquad \text{(V)}$$

This gives us the expected number of points within the cluster. But the actual number of points is α_i. Thus, we equally distribute the excess points among all the points constituting the cluster. This gives us an approximate value for MinPts (number of "close" neighbors) of the cluster:

$$\text{MinPts for cluster } i = \quad \gamma_i = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \qquad \text{(VI)}$$

After obtaining MinPts for all k clusters, we compute a weighted mean over all clusters to obtain the average number of MinPts for the entire sequence space.

$$\therefore MinPts = \left\lceil \frac{\sum_{i=1}^{k} \gamma_i \alpha_i}{\sum_{i=1}^{k} \alpha_i} \right\rceil \qquad \text{(VII)}$$

Only clusters with at least 2 points are used in this computation.

(c) Considering duplicates: But this approach gives a reasonable value for the average number of MinPts in a sequence space if all the points are unique and there are no duplicates. In our case, there are many instances when the sequences are exactly the same. This is representative of exactly same application behavior. Density is the basis of our system and we cannot ignore duplicates. But the equation (V) would be affected since the maximum distance still remains the same whereas the average value is suppressed due to the presence of points with same spatial coordinates. Also, if there are q points corresponding to a coordinate (x, y), then each of the q points is bound to have (q-1) MinPts in the worst case.

Let p be the number of frequent data points (i.e. frequency > 1) in cluster i. Let ψ_j be the frequency of a data point j in cluster i. We compute γ` the same way as equation (VI), where γ` is the MinPts value for cluster i assuming unique points in the sequence space.

$$\gamma'_i = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \qquad \text{(VIII)}$$

This value is then modified to accommodate the frequently occurring points (corresponding to sequences sharing the same spatial positions in the sequence space). We compute a weighted mean to obtain an appropriate value of MinPts in cluster i as follows:

$$\gamma_i = \left\lceil \frac{\gamma'_i \, \alpha_i + \sum_{j=1}^{p} \psi_j (\psi_j - 1)}{\alpha_i + \sum_{j=1}^{p} \psi_j} \right\rceil \qquad \text{(IX)}$$

Average MinPts for the entire plot can then be computed using equation (VII) above.

3.5.1.2 Threshold for raising alarms

LOF only assigns a local outlier factor for a point in the sequence space which corresponds to its anomaly score. If the score is above a user specified threshold, then it is considered as anomalous and hence filtered from the data set. If the threshold is too low, there is a risk of filtering a lot of points, many of which may depict normal application behavior. On the contrary, if the threshold is too high, some of the data points corresponding to actual intrusions (but close to many other data points on the sequence space) may not get filtered. One way to compute a threshold automatically is to order and plot the LOF scores in increasing order of the scores (with each data point along the X-axis and the anomaly/LOF score along the Y-axis). Since the normal points are assumed in abundance, their LOF scores are ideally 1. We are interested in the scores after the first steep rise of this plot, since these correspond to outliers. Ignoring all the scores below the first steep rise (corresponding to normal sequences), the cut-off value can be computed as the median of all the scores thereafter. This heuristic gives a reasonable threshold value for the various applications in our data sets.

3.6 Phase 6: Training and online detection

The filtered data set obtained above provides clean data as training input to any online anomaly detection system like stide and LERAD.

stide (Sequence TIme-Delay Embedding) [35] memorizes all contiguous sequences of predetermined, fixed length (n-grams) during training. This is done by using a sliding window and adding all unique sequences to the database. During test phase, an anomaly count is associated with n-gram mismatches and is defined as the number of mismatches in a temporally local region for a sequence. A threshold is set for the anomaly score above which a sequence is flagged anomalous, indicating a possible attack.

LERAD (LEarning Rules for Anomaly Detection) [24] is a randomized algorithm that learns rules for the normal data set. With every rule a probability is assigned for encountering a novel value of the attribute in the consequent when the conditions in the antecedent are true. A non-stationary model is assumed for LERAD – frequency is made irrelevant and only the last occurrence of an event is assumed important. The anomaly scoring function uses the probability and time since last anomaly of the rule violated by the test input. Details for the rule learning algorithm are available in [24]. The applicability of LERAD to host based anomaly detection has been demonstrated in [33].

Table 1. Automated MinPts computation

Application	eject	fdformat	ftpd	ps	lpr	login	excel
Number of sequences	21	19	91	341	3704	16	38
Average sequence length	66.43	57.63	284.93	66.45	835.73	730.81	2862.87
MinPts	3	2	10	71	6	3	2

4. EXPERIMENTAL EVALUATION

Our goal is to determine if our proposed representation can be used with an unsupervised learning algorithm (namely LOF) to detect and purge out anomalies, creating a "clean" training set for online detection systems. We would also like to note the change in performance after using our filtering scheme.

4.1 Data sets and preprocessing

We evaluated our techniques on 7 applications obtained from three different data sets:

(i) The DARPA intrusion detection evaluation data set was developed at the MIT-Lincoln Labs [22]. We used the Solaris data from the Basic Security Module (BSM) audit logs [26]. This data has to be preprocessed before use by MORPHEUS.

For our experiments, we selected the ftpd, ps, fdformat and eject applications to obtain a good range in the number of sequences and the number of system calls (~1200-26000). These applications also have a good mix of different attack types [15]. The ftpd application comprises of R2L (guessftp, ftpwrite) and DoS (warez, warezclient) attacks. On the other hand, ps, eject and fdformat are all U2R attacks.

(ii) Two applications (lpr and login) from the University of New Mexico (UNM) data sets [35] were also used. The lpr application comprised of 2703 normal traces running lpr collected from 77 hosts running SUNOS 4.1.4 at the MIT Artificial Intelligence Lab. Another 1001 traces correspond to the execution of the lprcp attack script. Older versions of `lpr` use only 1000 different names for printer queue files. The attack takes advantage of the fact that the old files are not removed from the queue before they can be used again. The attack works as follows: a symbolic link is placed to the victim file at the beginning. All the intermediary traces increment the counter and the intruder overwrites the target file in the last trace. Traces from the login application were obtained from a Linux machine running kernel 2.0.35. A homegrown Trojan program was used for the attack traces.

(iii) We also used system call sequence logs corresponding to Microsoft excel macros in execution used by researchers at Florida Institute of Technology (FIT) [38] and University of Tennessee at Knoxville (UTK) [25]. 36 normal traces correspond to some statistical, chemistry and cost estimation related Excel macros. 2 malicious traces modify the registry settings and execute some other application. Such a behavior is exhibited by the ILOVEYOU worm which opens the web browser to a specified website and executes a program, modifying registry keys and corrupting user files. This worm results in a distributed denial of service (DDoS) attack.

4.2 Outlier (anomaly) detection in sequence space

Our system creates a sequence space and plots all sequences as points with respect to other sequences. We claim that the malicious sequences are reflected as outliers in the sequence space. It is therefore imperative for us to evaluate if the outliers in the sequence space correspond to actual attacks. The underlying assumption is that the bulk of the data set constitutes of normal SCSs. We assume that the similar nature of normal behavior will cause them to cluster together. Outliers to these clusters would be the anomalies resulting from possible intrusions.

For the MIT-Lincoln lab data set, week 3 comprises of clean data while weeks 4 and 5 data has attacks. We are also given the timestamp for the occurrence of the attacks. In this experiment, after dividing the data into different applications and their processes (as explained in Section 4.1), we combine the data for the 3 weeks together (on a per application basis) and feed it to our system. This gives a good mix of normal application behavior and some sequences resulting from intrusions. We also use an aggregation of all traces for the other data sets (UNM and FIT-UTK) on a per application basis for similar reasons.

We created a sequence space for each application. Figure 3(a) represents the sequence space for the ftpd application from the DARPA evaluation data set, whereas 3(b) corresponds to the lpr data set from UNM. The X-axis on the plots is the distance due to the motif separation amongst sequences and Y-axis corresponds to the distance with respect to the motifs present in the sequence. Similar sequences tend to cluster together while anomalous sequences are represented as outliers.

We used the sequence space to detect local outliers using LOF on all the datasets. LOF takes MinPts-nearest neighbors (the number of points comprising the neighborhood of a point) as an input parameter and the results are very sensitive to this parameter selection [4]. For our experiments, we varied this parameter value as a percentage of the entire population. We also used the MinPts value that we computed using our automated technique (as in Section 3.5.1.1). These values are listed in Table 1. After computing the LOF or anomaly scores, we ranked them in descending order. All the sequences with scores greater than the threshold were considered anomalies and evaluated for detections and false alarms.

The results from the experiments, depicted in Table 2, indicate that none of the MinPts values were ideal to detect all the attacks. The two parameter values – 15% and 20% – seem to have the maximum number of detections (17 each, out of 19 total attacks).

Table 2. True positives and false positives for various applications at varied LOF MinPts values

Application	Total Attacks	Number of different attacks detected (with false alarm count) for different values of LOF MinPts (% of total population)				
		5%	**10%**	**15%**	**20%**	**Automated (from Table 1)**
eject	2	1 (1)	2 (1)	2 (0)	2 (0)	2 (0)
fdformat	3	3 (0)	3 (0)	3 (0)	3 (0)	3 (0)
ftpd	6	0 (6)	0 (11)	6 (6)	6 (1)	0 (11)
ps	4	0 (6)	4 (1)	4 (1)	4 (2)	4 (49)
lpr	1	0 (123)	1 (193)	1 (198)	1 (157)	1 (97)
login	1	0 (1)	0 (2)	1 (2)	1 (2)	1 (2)
excel	2	2 (0)	0 (3)	0 (0)	0 (0)	2 (0)
Total	**19**	**6 (137)**	**10 (211)**	**17 (207)**	**17 (162)**	**13 (159)**

The only attacks missed were the ones in the excel application where a reasonable value of MinPts is best suggested as 5%. Our methodology for MinPts calculation was successful in computing the correct number for the parameter and hence successfully detected the attack sequence as outlier (for which the 15% and 20% values failed). The automated LOF parameter detected all the attacks except the ones in the ftpd application. The reason for such a behavior can be better understood from Figure 3(a). There are 2 main clusters in the sequence space – one close to the origin and the other towards the center of the plot. The total number of points is 91 (80 in the large cluster, 10 in the smaller one, and one spurious point far away on the top-right corner of the plot). The MinPts value obtained by using our heuristic is 10, which seems to be an appropriate value. Inability of LOF to detect the anomalies in this representation is attributed to the fact that all the 10 points in the smaller cluster correspond to 6 different attacks. Therefore, the anomaly scores for all these points are very low. This implies that the concept of local outliers is not sufficient to capture such anomalous data points. Thus, we need to adopt a global view to find anomalous clusters as well, which can be incorporated in our sequence space. This would also be beneficial in detecting flooding attacks, which would typically correspond to high density points/clusters in the sequence space. Other than the ftpd application, the automated technique successfully detected all other attacks. This suggests that the MinPts values computed using our heuristic are generally reasonable.

As can be observed from Table 2, the number of false alarms is high for the lpr application (in the range 2.6 – 5.3%), which constitutes of over 3700 sequences and approximately 3.1 million system calls. The data was collected over 77 different hosts and represents high variance in application behavior. Though we were able to capture the lpr attack invoked by the lprcp attack script, we also detected other behavioral anomalies which do not correspond to attacks. Our goal here is to retain generic application behavior and shun anomalies. Peculiar (but normal) sequences would also be deemed anomalous since they are not representative of the general way in which the application functions, as in this case. Since program faults and system crashes are not representative of normal behavior, purging these anomalies is justifiable.

Our representation scheme also subsumes the ideas presented in [35, 36, 37, and 13]. The underlying assumption is that similar sequences would appear together in the sequence space. An attack modifies the course of events. This results in (a) either the absence of a motif, or (b) altered spatial positions of motifs within the sequence due to repetition of a motif, or (c) the presence of an entirely new motif. All these instances affect the spatial relationships amongst the different motifs within the sequence. Ultimately, this affects the distance of the malicious sequence with respect to the reference sequence, resulting in an outlier being plotted on the sequence space. It is this drift within the sequence space that the outlier detection algorithm is able to capture as an anomaly. Since the reference sequence is picked randomly, it may so happen that the reference sequence is the attack sequence itself. This does not affect our system since our distance metric is symmetric and the point is still classified as an outlier.

4.3 Effects of filtering the data

We reemphasize that our ultimate goal is to obtain "clean" data for other intrusion detection systems to train on. Thus, it is important to study how well our system can clean the training data and what effect does it have on the performance of an online detection system (in terms of true detections as well as false alarm generation).

Only the MIT-Lincoln Labs and FIT-UTK data sets were used for this set of experiments since they contained sufficient attacks to be used in both "adulterated" training and test data sets. The lpr and login data sets from UNM comprised of only a single attack. We have already demonstrated in the previous section that our technique could filter them as spurious outliers. But a single attack is not sufficient for this set of experiments, as we would like to have attacks in both training and test data. Therefore, we did not involve those two applications for this experiment. We combined the "clean" week 3 data with the "mixed" week 4 data of the MIT-Lincoln lab data set to obtain an unlabeled data set. We use this to train stide and LERAD. We then tested on week 5 data (containing attacks with known timestamps). Subsequently, we used MORPHEUS to filter out spurious data points (and hence SCSs) from the combined data set. This marks the end of phase 5 of our system. The sixth and final phase is next, which uses the refined data set for training stide and LERAD. Week 5 data is used for testing purposes. As per the 1999 DARPA evaluation criteria, a system is considered to have successfully detected an

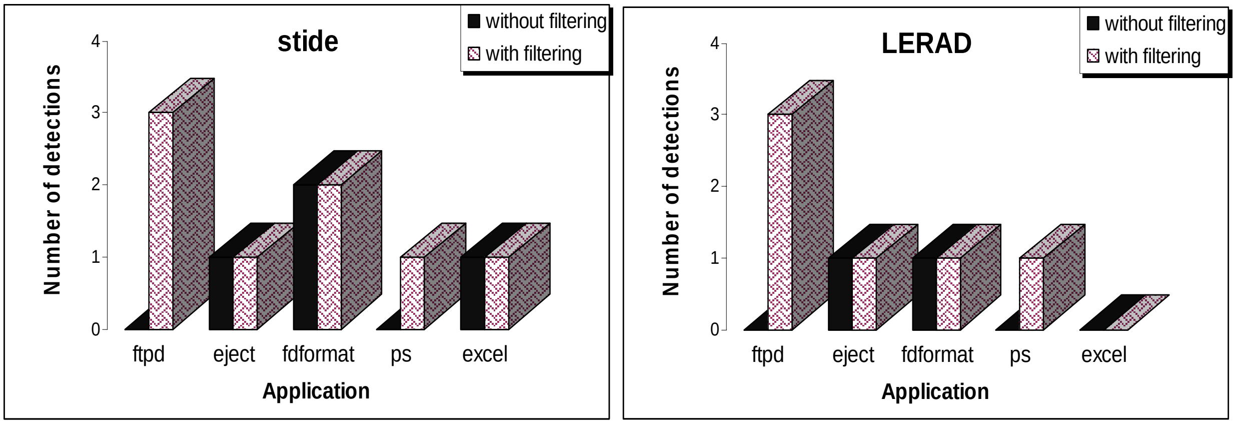


Figure 4. Comparison of attack detections with and without filtering (for stide and LERAD respectively).

attack if it generates an alarm within 60 seconds of the occurrence of the attack. We follow the same criterion for our evaluation. For the FIT-UTK Microsoft Excel data set, we randomly picked 33 traces (including one attack) for training and remaining 5 traces for testing purposes.

The parameter selection for our experiments was as follows: For stide, we used a window size of 6. A locality frame of 20 is used, that is the anomaly count keeps track of the number of mismatches in a temporally local region comprising 20 system calls. All the parameter values used are suggested to give best results in [35]; parameter sensitivity is studied in [32]. For LERAD, each tuple comprised of the system call, its return value and error status besides other arguments. In all cases, alarms are accumulated for the applications and then evaluated for the number of true detections and false positives.

Figure 4 depicts the number of attacks detected by stide and LERAD for the 5 applications under study. It is observed that in both cases, the IDS was able to detect more attacks in ftpd and ps applications after data filtering by MORPHEUS while there was no change in the performance for the other three applications (eject, fdformat and excel). This is because the training data also contained some attacks. For the ftpd and ps applications, the attacks in the test data were similar to the ones seen in the adulterated training data set, and were hence missed by both the IDSs. When filtered using MORPHEUS, the attack SCSs were purged and hence detections were possible in the test phase. For the applications eject and fdformat, the attacks in the adulterated training and testing data sets were different in character. Hence both the systems detected them irrespective of the filtering procedure. For excel, stide was able to detect the worm in both cases due to similar reasons. LERAD was not able to capture the malicious sequence due to incomplete argument information.

No false alarms were generated in any case in stide except the excel application, where one false positive was produced in each case. For LERAD, 1 false alarm was generated for the ps application with and without filtering. No other false positives were obtained. Overall, the results indicate that the filtering process was instrumental in increasing the number of detections without increasing the number of false alarms.

5. CONCLUSIONS AND FUTURE WORK

Most of the traditional host based IDSs require a “clean” training data set which is difficult to obtain. Our system, called MORPHEUS, addresses and attempts to solve this problem of data filtering. We present a motif-based representation for system call sequences (SCSs) based upon their spatial positions within the sequence. We also propose a novel representation of sequences – called sequence space – using a distance metric between the motif-based representations. We also exhibited the efficacy of this feature space to filter anomalies by integrating it with an existing unsupervised learning algorithm (called LOF) for outlier detection. Experiments were performed on different applications which varied in size, operating system (SUNOS, Solaris, Linux and Windows), and environment (simulated and live/real). Results indicate that our system can successfully detect the vulnerability-based anomalies. The generation of false alarms is caused by the irregularities in the data set and the results are sensitive to the parameter selection for the outlier algorithm. We proposed heuristics to automate the parameters to MORPHEUS – MinPts (a parameter to LOF) and threshold for raising alarms, thereby making our system parameter-free. Our automatically computed parameter was generally able to detect the attacks producing the least false alarms in the most irregular real data set. After filtering the anomalous points, the “clean” training data set was used by an online detection system resulting in higher detection rates, implying that MORPHEUS effectively purged the anomalies to create a better training data set.

An attacker might devise a clever technique to evade typical sequence-based anomaly detection systems. Wagner and Soto [34] presented one such idea wherein they were successful in modeling a malicious sequence by adding null operators to make it consistent with the sequence of system calls. The sequence based techniques dealing with short sub-string patterns can be bypassed by spreading the attack over longer duration (or longer subsequences). MORPHEUS uses variable length motifs and also takes the relative positions of the motifs for anomaly detection, and is better equipped and more robust against such evasions. In essence, our system models sequences at two different levels – at the individual motif level and also at the level of spatial relationship between motifs within the audit sequence. The latter level adds to the security of the system and would make it even harder for the attacker to evade the system, since he has to now not only use the “normal” audit event patterns, but also place those event-sequences/motifs within the respective sequence at proper relative positions.

MORPHEUS can be integrated with a hybrid of signature and anomaly based systems for better accuracy and the ability to detect novel attacks. Our system can also be used for user profiling and detecting masquerade. Also, as mentioned earlier, we need to expand our perspective from local to global outliers to detect attack clusters. In terms of efficiency, the only bottleneck in our system is the motif extraction phase where cross-match is performed pair-wise. Speed-up is possible by using other techniques like suffix trees. We are also working on refining the motif relationships in the motif-based representation.

6. REFERENCES

[1] C. Aggarwal and P. Yu. *Outlier Detection for High Dimensional Data*. SIGMOD, 2001.

[2] J. Allen. *Maintaining knowledge about temporal intervals*. Communications of the ACM 26, 11, 832-843, 1983.

[3] M. Bernaschi, E. Gabrielli and L.V. Mancini. *Operating System Enhancement to Prevent the Misuse of System Calls*. ACM CCS, 2001.

[4] M. Breunig, H. Kriegel, R. Ng and J. Sander. *LOF: Identifying Density-Based Local Outliers*. SIGMOD, pp. 93-104, 2000.

[5] P. Chan, M. Mahoney and M. Arshad. *Learning Rules and Clusters for Anomaly Detection in Network Traffic*. Managing Cyber Threats: Issues, Approaches and Challenges, V. Kumar, J. Srivastava and A. Lazarevic (editors), Kluwer, 2003.

[6] W. Cohen. *Fast Effective Rule Induction*. ICML, 1995.

[7] S. Coull, J. Branch, B. Szymanski and E. Breimer. *Intrusion Detection: A Bioinformatics Approach*. ACSAC, 2003.

[8] D.E. Denning, *An Intrusion Detection Model*, IEEE Transactions on Software Engineering, SE-13:222-232, 1987.

[9] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo. *A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data*. In D. Barbara and S. Jajodia (editors), Applications of Data Mining in Computer Security, Kluwer, 2002.

[10] S. Forrest, S. Hofmeyr, A. Somayaji and T. Longstaff. *A Sense of Self for UNIX Processes*. IEEE S&P, 1996.

[11] A. Ghosh and A. Schwartzbard. *A Study in Using Neural Networks for Anomaly and Misuse Detection*. USENIX Security Symposium, 1999.

[12] A.J. Gibbs and G.A. McIntyre. *The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences*. Eur. J. Biochem. 16:1-11, 1970.

[13] N. Jiang, K. Hua and S. Sheu. *Considering Both Intra-pattern and Inter-pattern Anomalies in Intrusion Detection*. ICDM, 2002.

[14] A. Jones and S. Li. *Temporal Signatures for Intrusion Detection*. ACSAC, 2001.

[15] K. Kendell. *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems*. Masters Thesis, MIT 1999.

[16] E. Knorr and R. Ng. *Algorithms for Mining Distance-based Outliers in Large Data Sets*. VLDB, 1998.

[17] T. Lane and C.E. Brodley. *Detecting the abnormal: Machine Learning in Computer Security*. TR-ECE 97-1, Purdue University, 1997.

[18] T. Lane and C.E. Brodley. *Sequence Matching and Learning in Anomaly Detection for Computer Security*. AI Approaches to Fraud Detection and Risk Management, 1997.

[19] A. Lazarevic, L. Ertoz, A. Ozgur, J. Srivastava and V. Kumar. *A comparative study of anomaly detection schemes in network intrusion detection*, SDM, 2003.

[20] W. Lee, S. Stolfo and P. Chan. *Learning Patterns from UNIX Process Execution Traces for Intrusion Detection*. Workshop, AI Approaches to Fraud Detection and Risk Management, 1997.

[21] Y. Liao and R. Vemuri. *Use of Text Categorization Techniques for Intrusion Detection*, 11th USENIX Security Symposium, 2002.

[22] R. Lippmann, J. Haines, D. Fried, J. Korba and K. Das. *The 1999 DARPA Off-Line Intrusion Detection Evaluation*. Computer Networks (34) 579-595, 2000.

[23] P.C. Mahalanobis. *On Tests and Measures of Groups Divergence*. International Journal of the Asiatic Society, Vol. 26:541, 1930.

[24] M. Mahoney and P. Chan. Learning *Rules for Anomaly Detection of Hostile Network Traffic*, ICDM, 2003.

[25] G. Mazeroff, Victor De Cerqueira, J. Gregor and M.G. Thomason. *Probabilistic Trees and Automata for Application Behavior Modeling*. 41st ACM Southeast Regional Conference Proceedings, 2003.

[26] W. Osser and A. Noordergraaf. *Auditing in the Solaris 8 Operating Environment*. Sun Blueprints Online.

[27] L. Portnoy. *Intrusion Detection with Unlabeled Data Using Clustering*, Undergraduate Thesis, Columbia University, 2000.

[28] S. Ramaswamy, R. Rastogi and K. Shim, *Efficient Algorithms for Mining Outliers from Large Data Sets*, Proceedings of the ACM SIGMOD Conference, 2000.

[29] I. Rigoutsos and A. Floratos. *Combinatorial pattern discovery in biological sequences*. Bioinformatics, 14(1):55–67, 1998.

[30] S. Salvador, P. Chan and J. Brodie. *Learning States and Rules for Time Series Anomaly Detection*. FLAIRS, 2004.

[31] R. Sekar, M. Bendre, D. Dhurjati and P. Bollineni. *A Fast Automaton-based Method for Detecting Anomalous Program Behaviors*. IEEE S&P, 2001.

[32] K. Tan & R. Maxion. *"Why 6?" Defining the Operational Limits of stide*. IEEE S&P, 2002.

[33] G. Tandon and P. Chan. *Learning Rules from System Call Arguments and Sequences for Anomaly Detection*. DMSEC, 2003.

[34] D. Wagner and P. Soto. *Mimicry Attacks on Host-Based Intrusion Detection Systems*. ACM CCS, 2002.

[35] C. Warrender, S. Forrest and B. Pearlmutter. *Detecting Intrusions Using System Calls: Alternative Data Models*. IEEE S&P, 1999.

[36] A. Wespi, M. Dacier and H. Debar. *An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm*. Proc. EICAR, 1999.

[37] A. Wespi, M. Dacier and H. Debar. *Intrusion detection using variable-length audit trail patterns*. RAID, 2000.

[38] J.A. Whittaker and A. De Vivanco. *Neutralizing Windows-based malicious mobile code*. ACM Symposium on Applied Computing, 2002.