Data Cleaning and Enriched Representations for Anomaly Detection in System Calls

Gaurav Tandon, Philip Chan and Debasis Mitra

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{gtandon, pkc, dmitra}@cs.fit.edu

1 Introduction

Computer security research has two major aspects: intrusion prevention and intrusion detection. While the former deals with preventing the occurrence of an attack (using authentication and encryption techniques), the latter focuses on the detection of successful breach of security. Together, these complementary approaches assist in creating a more secure system.

Intrusion detection systems (IDSs) are generally categorized as misuse-based and anomaly-based. In misuse (signature) detection, systems are modeled upon known attack patterns and the test data is checked for occurrence of these patterns. Examples of signature-based systems include virus detectors that use known virus signatures and alert the user when the system has been infected by the same virus. Such systems have a high degree of accuracy but suffer from the inability to detect novel attacks. Anomaly based intrusion detection [1] 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. Intrusion detection systems can also be categorized as network-based, which monitors network traffic, and host-based, where operating system events are monitored.

There are two focal issues that need to be addressed for a host-based anomaly detection system: cleaning the training data, and devising an enriched representation for the model(s). Both these issues try to improve the performance of an anomaly detection system in their own ways. First, all the proposed techniques that monitor system call sequences rely on *clean* training data to build their model. Current audit sequence is then examined for anomalous behavior using some supervised learning algorithm. 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. Moreover, obtaining clean data by hand could be tedious. Purging all malicious content from audit data using an automated technique is hence imperative.

Second, normal behavior has to be modeled using features extracted from the training set. It is important to remember that the concept of normalcy/abnormality in anomaly detection is vague as compared to a virus detector which has an exact signature of the virus it is trying to detect, making anomaly detection a hard problem. Traditional host-based anomaly detection systems focus on system call sequences to build models of normal application behavior. These techniques are based upon the observation that a malicious activity results in an abnormal (novel) sequence of system calls. Recent research [2, 3] has shown that sequence-based systems can be compromised by conducting mimicry attacks. Such attacks are possible by astute execution of the exploit by inserting dummy system calls with invalid arguments such that they form a legitimate sequence of events, thereby evading the IDS. A drawback of sequence-based approaches lies in their non-utilization of other key attributes, namely the system call arguments. The efficacy of such systems might be improved upon if a richer set of attributes (return value, error status and other arguments) associated with a system call is used to create the model.

In this chapter, we address the issues of data cleaning and anomaly detection, both of which essentially try to detect outliers, but differ in character:

1. Offline vs. online techniques. We present two enriched representations: (i) motifs and their locations are used for cleaning the data (an offline procedure) whereas (ii) system call arguments are modeled for online anomaly detection.
2. Supervised algorithms assume no attacks in the training data. Unsupervised algorithms, on the other hand, relax this constraint and could have small amounts of unlabeled attacks. We present two modified detection algorithms: Local Outlier Factor or LOF (unsupervised) and LEarning Rules for Anomaly Detection or LERAD (supervised).
3. Low false alarm rates are critical in anomaly detection and desirable in data cleaning. False alarms are generated in anomaly detection systems as not all anomalies are representative of attacks. For an online detection system, a human expert has to deal with the false alarms and this could be overwhelming if in excess. But for data cleaning, we would like to retain generic application behavior to provide a clean data set. Rendering the data free of attacks is highly critical and some other non-attack abnormalities may also get removed in the process. Purging such anomalies (program faults, system crashes among others) is hence justifiable, if still within reasonable limits. Thus the evaluation criteria vary in the two cases.

Empirical results indicate that our technique is effective in purging anomalies in unlabeled data. Our representation can be effectively used to detect malicious sequences from the data using unsupervised learning techniques. The filtered training data leads to better application modeling and an enhanced performance (in terms of the number of detections) for online anomaly detection systems. Our system does not depend on the user for any parameter values, as is the norm for most of the anomaly detection systems. Our sequence and argument based representations also result in better application modeling and help detect more attacks than the conventional sequence-based techniques.

This chapter is organized as follows. Sect. 2 reviews some prevalent anomaly detection systems. In Sect. 3 we present the concept of motifs (in the context of system call sequences) and motif-based representations for data cleaning. Sect. 4 presents the argument-based representations and supervised learning algorithm for anomaly detection. An experimental evaluation is presented in Sect. 5 and we conclude in Sect. 6.

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. [4] memorized normal system call sequences using a look-ahead pairs (tide). Lane and Brodley [5, 6] examined UNIX command sequences to capture normal user profiles using a fixed size window. Later work (stide and t-stide) by Warrender et al. [7] extended sequence modeling by using n-grams and their frequency. Wespi et al. [8, 9] proposed a scheme with variable length patterns using Teiresias [10], a pattern discovery algorithm in biological sequences. Ghosh and Schwartzbard [11] used artificial neural networks, Sekar et al. [12] proposed a finite state automaton, Jiang et al. [13] also proposed variable length patterns, Liao and Vemuri [14] used text categorization techniques, Jones and Li [15] learnt temporal signatures, Coull et al. [16] suggested sequence alignment, Mazeroff et al. [17] proposed probabilistic suffix trees, and Lee at al. [18] used a machine learning algorithm called RIPPER [19] 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 [20, 21, 22, 23]. Network traffic comprises 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.

From the modeling/detection point of view, all the above mentioned approaches for host-based systems use system call sequences. Parameter effectiveness for window based techniques has been studied in [3]. Given some knowledge about the system being used, attackers can devise some methodologies to evade such intrusion detection systems. Wagner and Soto [2] modeled a malicious sequence by adding *no-ops* (system calls having no effect) to compromise an IDS based upon the sequence of system calls. Such attacks would be detected if the system call arguments are also taken into consideration, and this provides the motivation for our work.

3 Data Cleaning

The goal is to represent sequences in a single feature space and refine the data set offline by purging anomalies using an unsupervised learning technique on the feature space.

3.1 Representation with motifs and their locations

Let Σ be a finite set of all distinct system calls. A system call sequence (*SCS*) s is defined as a finite sequence of system calls and is represented as (c_1 c_2 c_3 ... c_n), where $c_i \in \Sigma, 1 \leq i \leq n$.

After processing the audit data into process executions, system call sequences are obtained as finite length strings. Each system call is then mapped to a unique symbol using a translation table. Thereafter, they are ranked by utilizing prior knowledge as to how susceptible the system call is to malicious usage. A ranking scheme similar to the one proposed by Bernaschi et al. [24] was used to classify system calls on the basis of their threat levels.

Motifs and motif extraction

A *motif* is defined as a subsequence of length greater than p if it appears more than k times, for positive integers p and k, within the finite set $S = \{s_1, s_2, \ldots, s_m\}$ comprising m *SCS*s. Motif discovery has been an active area of research in bioinformatics, where interesting patterns in amino and nucleic acid sequences are studied. Since motifs provide a higher level of abstraction than individual system calls, they are important in modeling system call sequences. Two sets of motifs are extracted via *auto-match* and *cross-match*, explained next.

The set of motifs obtained through auto-match comprise 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 *SCS*s S is the input to this algorithm, a set of unique motifs $M = \{m_1, m_2, \ldots, m_q\}$ is the output. It may happen that a shorter 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. More formally, a motif 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

$$acggcggfgjcggfgjxyz \tag{1}$$

Note that in this sequence we have a motif *cgg* with frequency 3, and another motif *cggf* with frequency 2, which is longer and sometimes subsumes the shorter motif but not always. We consider them as two different motifs since the frequency of the shorter motif was higher than the longer one. 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*}.

Apart from frequently occurring patterns, we are also interested in patterns which do not occur frequently but are present in more than one *SCS*. These motifs could be instrumental in modeling an intrusion detection system since they reflect common behavioral patterns across sequences (benign as well as intrusive). We performed pair-wise cross-match between different sequences to obtain these. In other words, 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:

$$acgfgjcgfgjxyzcg \tag{2}$$

$$cgfgjpqrxyzpqr \tag{3}$$

Using cross-match between the example sequences (2) and (3), 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 $MatrixMethod$ is described by Gibbs and McIntyre [25]. 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 between the two sequences. Using a technique similar to the Matrix Method, motifs are extracted which occur across sequences but may not be frequent within a single sequence itself.

Motifs obtained for a sequence (auto-match) or pairs of sequences (cross-match) are added to the motif database. Redundant motifs are removed. Motifs are then ordered based upon the likelihood of being involved in an attack.

The ranking for individual system calls is used here and motifs are ordered using dictionary sort. The motifs are then assigned a unique id based upon their position within the ordered motif database.

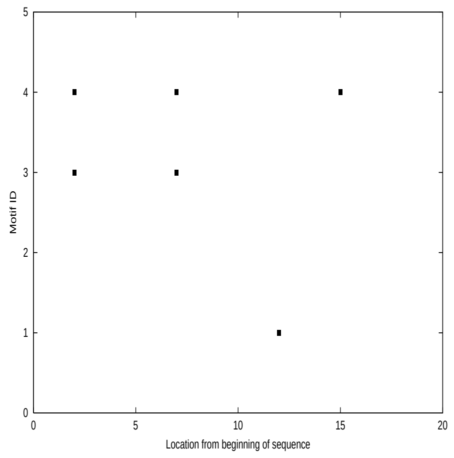

Fig. 1. Motif-oriented representation for sequence (2)

Motif-based representation of a sequence

After collecting 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 occurring 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. A sequence can thus be visualized as a scatter plot of the motifs present in the sequence. Fig. 1 depicts such a representation for the synthetic sequence (2), where the motifs *cg*, *cgfgj* and *xyz* are represented at the positions of occurrence within the respective sequence. A total of 4 unique motifs (*cg*, *cgfgj*, *pqr* and *xyz*), obtained from auto-match and cross-match of sequences (2) and (3), are assumed in the motif database for the plot in Fig. 1. 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.

All the *SCS*s are modeled based upon the contained 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 *SCS*s (based upon their motif-based representations) in a single feature space could reflect the similarity/dissimilarity between them.

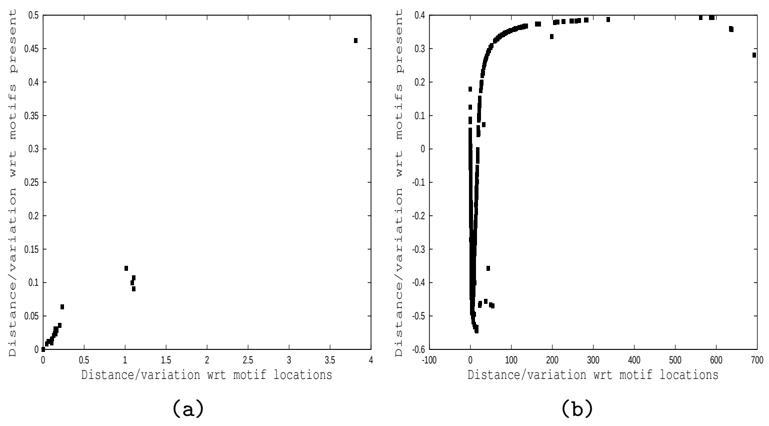

Fig. 2. Sequence space for two applications (a) ftpd and (b) lpr

A single representation for multiple sequences

After creating a motif-based representation for each sequence, all the test sequences S are plotted in a feature space called the *sequence space*. In this representation we measure the distance between pairs of *SCS*s 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 *SCS*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. Let x_{1_i} (x_{2_i}) be the position of the i^{th} motif in s_1 (s_2). Inspired by the symmetric Mahalanobis distance [26], the distance is computed as follows:

$$d_x = \frac{\frac{\sum_{i=1}^{n_1}(x_{1_i} - \bar{x}_2)}{\sigma_{x_2}} + \frac{\sum_{j=1}^{n_2}(x_{2_j} - \bar{x}_1)}{\sigma_{x_1}}}{n_1 + n_2} \quad d_y = \frac{\frac{\sum_{i=1}^{n_1}(y_{1_i} - \bar{y}_2)}{\sigma_{y_2}} + \frac{\sum_{j=1}^{n_2}(y_{2_j} - \bar{y}_1)}{\sigma_{y_1}}}{n_1 + n_2} \quad (4)$$

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, and $(\bar{x}, \bar{y})$ 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 Fig. 2. The origin represents the reference sequence. It is important to note that the position of another sequence (calculated using Eq. 4) 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 Fig. 2(b).

3.2 Unsupervised training with Local Outlier Factor (LOF)

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. In Fig. 2(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 a well researched topic in databases and knowledge discovery [27, 28, 29, 30]. It is important to note that an outlier algorithm with our representation is inappropriate for online detection since it requires complete knowledge of all process sequences.

LOF [28] is a density-based outlier finding algorithm which defines a local neighborhood, using which a degree of outlierness is assigned to every object. The number of neighbors ($MinPts$) is an input parameter to the algorithm. A reachability density is calculated for every object which is the inverse of the average reachability distance of the object from its nearest neighbors. 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 (anomaly score). Our system computes the anomaly scores for all the *SCS*s (represented as points in sequence space). All the points for which the score is greater than a threshold are considered anomalous and removed.

We made some modifications to the original LOF algorithm to suit our needs. In the original paper [28], all the points are considered to be unique and there are no duplicates. In our case, there are many instances when the sequences are exactly the same (representative of identical application behavior). The corresponding points would thus have the same spatial coordinates within the sequence space. Density is the basis of our system and hence we cannot ignore duplicates. Also, a human expert would be required to analyze the sequence space and suggest a reasonable value of $MinPts$. But the LOF values increase and decrease non-monotonically [28], making the automated selection of $MinPts$ highly desirable. We present some heuristics to automate the LOF and threshold parameters, making it a parameter-free technique.

Automating the parameters

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. We use the L-Method [31] 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 [31].

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, $1 \leq i \leq k$. 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

$$\beta_i = \left(\frac{\rho_i}{\tau_i}\right)^2 \tag{5}$$

This gives us the expected number of points within the cluster. But the actual number of points is a_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 i ($= \gamma_i$)

$$\gamma_i = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \tag{6}$$

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.

$$MinPts = \left\lceil \frac{\sum_{i=1}^{k} \gamma_i \alpha_i}{\sum_{i=1}^{k} \alpha_i} \right\rceil \tag{7}$$

Only clusters with at least 2 points are used in this computation. But this approach gives a reasonable value for the average number of $MinPts$ in a sequence space if all the points are unique. In case of duplicates, Eq. 5 is 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. If there are q points corresponding to a coordinate (x, y), then each of the q points is bound to have at least (q-1) $MinPts$.

Let p be the number of frequent data points (i.e. $frequency \geq 2$) in cluster i. Let ψ_j be the frequency of a data point j in cluster i. In other words, it is the number of times that the same instance occurs in the data. We compute γ' the same way as Eq. 6, where γ' is the $MinPts$ value for cluster i assuming unique points (no multiple instance of the same data point) in the sequence space.

$$\gamma'_i = \left\lceil \frac{\alpha_i - \beta_i}{\beta_i} \right\rceil \tag{8}$$

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:

$$\gamma_i = \left\lceil \frac{\gamma'_i \alpha_i + \sum_{j=1}^{p} \psi_j (\psi_j - 1)}{\alpha_i + \sum_{j=1}^{p} \psi_j} \right\rceil \tag{9}$$

Average $MinPts$ for the entire plot can then be computed using Eq. 7.

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. We present a heuristic to compute the threshold automatically without the need of any human expert. One point to note here is that the effect of false alarms for data purification is not as adverse as that of false alarm generation during online detection, if still within reasonable limits which are defined by the user. We compute the threshold automatically by ordering the LOF scores and plotting them in increasing order (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.

4 Anomaly Detection

The filtered data set obtained above can provide attack-free training input to any supervised learning algorithm that performs anomaly detection.

4.1 Representation with arguments

System call sequences have been effectively used in host based systems where a sliding window of fixed length is used. We introduce the term tuple to represent an instance of the sliding window. The simplest representation (denoted by *S-rep*) would therefore consist of 6 contiguous system call tokens (since length 6 is claimed to give best results in stide and t-stide [7]):

$$s_0\ s_1\ s_2\ s_3\ s_4\ s_5$$

where s_i is the system call at a distance i from the current system call within the 6-gram.

Consider the following sequence of system calls: *open, read, write, ... close.* This would seem like a perfectly normal sequence of events corresponding to a typical file access. But what happens if the file being accessed is *passwd*? Only the super-user should have the rights to make any modifications to this file. Malicious intent involving this file would not be captured if only the system call sequences are monitored. This lays stress on the enhancement of features to enrich the representation of application behavior at the operating system level. The enhancement for host-based systems, as is obvious from the example above, is to scrutinize the system call arguments as well.

The argument-based representation, denoted by *A-rep*, takes into consideration the various attributes like return value, error status, besides other arguments pertaining to the current system call. Let the maximum number of attributes for any system call be . The representation would now consist of tuples of the form:

$$s_0\ a_1\ a_2\ a_3\ ...\ a_\eta$$

where s_0 is the current system call are a_i is its i^{th} argument. It is important to note that the order of the attributes within the tuple is system call dependent. Also, by including all possible attributes associated with the system call, we can maximize the amount of information that can be extracted from the audit logs. We fix the total number of arguments in the tuple to the maximum number of attributes for any system call. If any system call does not have a particular attribute, it is replaced by a $NULL$ value.

S-rep models system call sequences whereas A-rep adds argument information, so merging the two representations is an obvious choice. The merged representation, called *M-rep*, comprises tuples containing all the system call within the 6-gram (S-rep) along with the attributes for the current system call (A-rep). The tuple is thus represented as

$$s_0\ a_1\ a_2\ a_3\ ...\ a_\eta\ s_1\ s_2\ s_3\ s_4\ s_5$$

A further modification to the feature space includes all the system calls in the fixed sliding window and the η attributes for all the system calls within that window. This enhanced representation is denoted as *M*-rep.*

4.2 Supervised training with LERAD

The efficacy of host-based anomaly detection systems might be enhanced by enforcing constraints over the system calls and their arguments. Due to the enormous size and varied nature of the applications, manual constraint formulation is a tedious task. Moreover, due the nature of the attributes it might not be feasible for even a group of human experts to develop a complete set of constraints in a short span of time. The workaround to this problem is to use a machine learning approach which can automatically learn the important associations between the various attributes without the intervention of a human expert. An important aspect of rule learning is the simplicity and comprehensibility of the rules. The solution can be formulated as a 5-tuple (A, $\varPhi$, I, $\Re$, ς), where A is the set of N attributes, $\varPhi$ is the set of all possible values for the attributes in A, I is the set of input tuples which is a subset of the N-ary Cartesian product over A, $\Re$ is the rule set, and ς is the maximum number of conditions in a rule.

LERAD is an efficient conditional rule-learning algorithm. A LERAD rule is of the form

$$(\alpha_i = \phi_p) \wedge (\alpha_j = \phi_q) \ldots \varsigma terms \Rightarrow \alpha_k \in \{\phi_a, \phi_b \ldots\}$$

where α_i, α_j, α_k are the attributes, and ϕ_p, ϕ_q, ϕ_a, ϕ_b are the values for the corresponding attributes. Algorithms for finding association rules (such as Apriori [32]) generate a large number of rules. But a large rule set would incur large overhead during the detection phase and may not be appropriate to attain our objective. We would like to have a *minimal* set of rules describing the normal training data. LERAD forms a small set of rules. It is briefly described here; more details can be obtained from [33].

For each rule in $\Re$ LERAD associates a probability p of observing a value not in the consequent:

$$p = \frac{r}{n} \tag{10}$$

where r is the cardinality of the set in the consequent and n is the number of tuples that satisfy the rule during training. This probability estimation of novel (zero frequency) events is due to Witten and Bell [34]. Since p estimates the probability of a novel event, the larger p is, the less anomalous a novel event is. During the detection phase, tuples that match the antecedent but not the consequent of a rule are considered anomalous and an anomaly score is associated with every rule violated. When a novel event is observed, the degree of anomaly (anomaly score) is estimated by:

$$AnomalyScore = \frac{1}{p} = \frac{n}{r} \tag{11}$$

A non-stationary model is assumed for LERAD since novel events are bursty in conjunction with attacks. A factor t is introduced, which is the time interval since the last novel (anomalous) event. When a novel event occurred recently (i.e. small value for t), a novel event is more likely to occur at the present moment. Hence the anomaly should be low. This factor is therefore multiplied to the anomaly score, modifying it to t/p. Since a record can deviate from the consequent of more than one rule, the total anomaly score of a record is aggregated over all the rules violated by the tuple to combine the effect from violation of multiple rules:

$$TotalAnomalyScore = \sum_i \left(\frac{t_i}{p_i}\right) = \sum_i \left(\frac{tn_i}{r_i}\right) \tag{12}$$

where i is the index of a rule which the tuple has violated. The anomaly score is aggregated over all the rules. The more the violations, more critical the anomaly is, and the higher the anomaly score should be. An alarm is raised if the total anomaly score is above a threshold.

We used the various feature representations discussed in Sect. 4.1 to build models per application using LERAD. We modified the rule generation procedure enforcing a stricter rule set. All the rules were forced to have system call in the antecedent since it is the key attribute in a host based system. The only exception we made was the generation of rules with no antecedent.

Sequence of system calls: S-LERAD

Before using argument information, it was important to know whether LERAD would be able to capture the correlations among system calls in a sequence. So we used the S-rep to learn rules of the form:

$$(s_0 = close) \wedge (s_1 = mmap) \wedge (s_5 = open) \Rightarrow s_2 \in \{munmap\}$$
$$(1/p = n/r = 455/1)$$

This rule is analogous to encountering close as the current system call (represented as s_0), followed by *mmap* and *munmap*, and *open* as the sixth system call (s_5) in a window of size 6 sliding across the audit trail. Each rule is associated with an n/r value. The number 455 in the numerator refers to the number of training instances that comply with the rule (n in Eq. 11). The number 1 in the denominator implies that there exists just one distinct value of the consequent (*munmap* in this case) when all the conditions in the premise hold true (r in Eq. 11).

Argument-based model: A-LERAD

We propose that argument and other key attribute information is integral to modeling a good host-based anomaly detection system. We used A-rep to generate rules. A sample rule is

$$(s_0 = munmap) \Rightarrow a_1 \in \{0x134, 0102, 0x211, 0x124\}$$
$$(1/\mathrm{p} = n/r = 500/4)$$

In the above rule, 500/4 refers to the n/r value (Eq. 11) for the rule, that is, the number of training instances complying with the rule (500 in this case) divided by the cardinality of the set of allowed values in the consequent. The rule in the above example is complied by 500 tuples and there are 4 distinct values for the first argument when the system call is *munmap*.

Merging system call sequence and argument information of the current system call: M-LERAD

A merged model (M-rep) is produced by concatenating S-rep and A-rep. Each tuple now consists of the system call, various attributes for the current system call, and the previous five system calls. The n/r values obtained from the all rules violated are aggregated into the anomaly score, which is then used to generate an alarm based upon the threshold. An example of an M-LERAD rule is

$$(s_0 = munmap) \wedge (s_5 = close) \wedge (a_3 = 0) \Rightarrow s_2 \in \{munmap\}$$
$$(1/p = n/r = 107/1)$$

Merging system call sequence and argument information for all system calls in the sequence: M*-LERAD

All the proposed variants, namely S-LERAD, A-LERAD and M-LERAD, consider a sequence of 6 system calls and/or take into the arguments for the current system call. We propose another variant called multiple argument LERAD (M*-LERAD) in addition to using the system call sequence and the arguments for the current system call, the tuples now also comprise the arguments for the other system calls within the fixed length sequence of size 6 (M*-rep).

Can a rule learning algorithm efficiently generalize over the various features extracted? Does extracting more features, namely arguments along with system call sequences, result in better application modeling? More specifically, does the violation of rule(s) relate to an attack? And would this result in an increase in attack detections while lowering the false alarm rate? These are some of the questions we seek answers for.

5 Experimental Evaluations

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

1. The DARPA intrusion detection evaluation data set was developed at the MIT-Lincoln Labs [35]. Various intrusion detection systems were evaluated using a test bed involving machines comprising Linux, SunOS, Sun Solaris and Windows NT systems. We used the BSM audit log for the Solaris host;
2. The University of New Mexico (UNM) data set has been used in [4, 7, 13]; and
3. FIT-UTK data set consists of excel macro executions [36, 17].

Table 1. Effect of LOF *MinPts* values

Application	*Total Attacks*	*Number of attacks detected (with false alarm count) for different values of MinPts (% of total population)*				
		5%	*10%*	*15%*	*20%*	*Automated*
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)**

5.1 Data cleaning

Evaluation Procedures and Criteria

From the DARPA/LL data set, we used data for ftpd, ps, eject and fdformat applications, chosen due to their varied sizes. We also expected to find a good mix of benign and malicious behavior in these applications which would help us to evaluate the effectiveness of our models. We used BSM data logged for weeks 3 (attack-free), 4 and 5 (with attacks and their timestamps). Two applications (lpr and login) from the UNM data set and excel logs from the FIT-UTK data set were used. LOF was used to detect outliers in the sequence space for all the applications. The number of true positives was noted for different $MinPts$ values 5%, 10%, 15%, 20% of the entire population, as well as the value obtained using our heuristic.

Results and Analysis

For various values of the $MinPts$ parameter to LOF, our technique was successfully able to detect all the 19 attacks in the data set, as depicted in Table 1. But no single value of $MinPts$ was ideal to detect all the attacks. The two parameter values 15% and 20% seem to have the maximum number of detections (17 each). 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 automated $MinPts$ 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. Inability of LOF to detect the anomalies in this representation is attributed to the fact that all the points in the cluster correspond to attacks. The automated technique successfully detected all other attacks, suggesting that the $MinPts$ values computed using our heuristic are generally reasonable.

The number of false alarms generated is very high for the lpr application, 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. We reiterate that our goal 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.

The reason why our representation plots attacks as outliers is as follows. An attack modifies the course of events, resulting 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.

5.2 Anomaly detection with arguments

Evaluation Procedures and Criteria

For the DARPA/LL data set, we used data for ftpd, telnetd, sendmail, tcsh, login, ps, eject, fdformat, sh, quota and ufsdump applications, chosen due to their varied sizes - ranging from very large (over 1 million audit events) to very small (1500 system calls). We used week 3 data for training and weeks 4 and 5 for testing. We also used data for three applications (lpr, login and ps) from the UNM data set and as well as logs from the FIT-UTK excel macro data set. The input tuples for S-LERAD, A-LERAD, M-LERAD, and M*-LERAD were as discussed in Sect. 4.2. For tide, stide, and t-stide we used a window size of 6. For all the techniques, alarms were merged in decreasing order of the anomaly scores and then evaluated for the number of true detections at varied false alarm rates.

Results and Analysis

When only sequence based techniques are compared, both S-LERAD and t-stide were able to detect all the attacks in UNM and FIT-UTK data sets. However, t-stide generated more false alarms for the ps and excel applications (58 and 92 respectively) as compared to 2 and 0 false alarms in the case of S-LERAD. For the DARPA/LL data set, tide, stide and t-stide detected the most attacks at zero false alarms, but are outperformed by S-LERAD as the threshold is relaxed (Table 2). It can also be observed from the table that

Table 2. Comparison of sequence and argument based representations

Technique	*Number of attacks detected at different false alarm rates ($\times 10^{-3}$ % false alarms)*				
	0.0	*0.25*	*0.5*	*1.0*	*2.5*
tide	5	6	6	8	9
stide	5	6	9	10	12
t-stide	5	6	9	10	13
S-LERAD	3	8	10	14	15
A-LERAD	3	10	13	17	19
M-LERAD	3	8	14	16	19
M*-LERAD	1	2	4	11	18

A-LERAD fared better than S-LERAD and the other sequence-based techniques, suggesting that argument information is more useful than sequence information. A-LERAD performance is similar to that of M-LERAD, implying that the sequence information is redundant; it does not add substantial information to what is already gathered from arguments. M*-LERAD performed the worst among all the techniques at false alarms rate lower than 0.5×10^{-3} %. The reason for such a performance is that M*-LERAD generated alarms for both sequence and argument based anomalies. An anomalous argument in one system call raised an alarm in six different tuples, leading to a higher false alarm rate. As the alarm threshold was relaxed, the detection rate improved. At the rate of 2.5×10^{-3} % false alarms, S-LERAD detected 15 attacks as compared to 19 detections by A-LERAD and M-LERAD, whereas M*-LERAD detected 18 attacks correctly.

The better performance of LERAD variants can be attributed to its anomaly scoring function. It associates a probabilistic score with every rule. Instead of a binary (present/absent) value (as in the case of stide and t-stide), this probability value is used to compute the degree of anomalousness. It also incorporates a parameter for the time elapsed since a novel value was seen for an attribute. The advantage is twofold: (i) it assists in detecting long term anomalies, and (ii) it suppresses the generation of multiple alarms for novel attribute values in a sudden burst of data. An interesting observation is that the sequence-based techniques generally detected the U2R attacks whereas the R2L and DoS attacks were better detected by the argument-based techniques.

Our argument-based techniques detected different types of anomalies. In most of the cases, the anomalies did not represent the true nature of the attack. Some attacks were detected by subsequent anomalous user behavior; few others were detected by learning only a portion of the attack, while others were detected by capturing intruder errors. Although these anomalies led to the detection of some attacks, some of them were also responsible for raising false alarms, a problem inherent to all anomaly detection systems.

Table 3. Effects of filtering the training data on the performance in test phase

Application	*Number of attacks detected (false alarms generated) - stide*		*Number of attacks detected (false alarms generated) - LERAD*	
	Without filtering	*With filtering*	*Without filtering*	*With filtering*
ftpd	0(0)	3(0)	0(0)	3(0)
eject	1(0)	1(0)	1(0)	1(0)
fdformat	2(0)	2(0)	1(0)	1(0)
ps	0(0)	1(0)	0(1)	1(1)

5.3 Anomaly detection with cleaned data vs. raw data

Evaluation Procedures and Criteria

Only the MIT-Lincoln Labs data set was used for this set of experiments since they contained sufficient attacks and argument information to be used in both *adulterated* training and test data sets. 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 filtered out the outliers detected from the combined data set. The refined data set was then used to train stide and LERAD. Week 5 data was used for testing purposes. The input to stide and LERAD was discussed in Sect. 5.2. In all cases, alarms are accumulated for the applications and then evaluated for the number of true detections and false positives. The results are displayed in Table 3.

Results and Analysis

For the four applications in the data set, stide was able to detect 3 attacks without filtering the training data compared to 7 attacks after refining the training set. For LERAD, there were 2 detections before filtering versus 6 true positives after purging the anomalies in training data. Thus, in both cases, there was an improvement in the performance of the system in terms of the number of attack detections. The better performance is attributed to the fact that the some of the attacks in the adulterated training and testing data sets were similar in character. With the adulterated training data, the attacks were assumed normal resulting in the creation of an improper model. Hence both the systems missed the attacks during the test phase. But our filtering procedure was able to remove the anomalies from the training data to create a correct model of normal application behavior. This led to the detection of the attacks during testing.

It can also be noted from the table that no false alarms were generated in any experiment with stide. For LERAD, there was only one false alarm for the ps application (with and without filtering). Our results indicate that the filtering mechanism was effective in purging out anomalies in training data, resulting in the detection of more attacks without increasing the number of false alarms in the test phase.

6 Concluding Remarks

In this chapter, we present two enriched representations: (i) motifs and their locations are used to represent sequences in a single sequence space, this being an offline procedure, and (ii) system call arguments modeled for online anomaly detection. We also presented two different aspects of machine learning: unsupervised learning and supervised learning for anomaly detection. Our techniques cater to the various issues and can be integrated to form a complete host-based system.

Most of the traditional host based IDSs require a *clean* training data set which is difficult to obtain. We present a motif-based representation for system call sequences (*SCS*s) based upon their spatial positions within the sequence. Our system also creates a single representation for all *SCS*s called sequence space - using a distance metric between the motif-based representations. A local outlier algorithm is used to purge this data void of all attacks and other anomalies. We demonstrate empirically that this technique can be effectively used to create a *clean* training data set. This data can then be used by any supervised anomaly detection system to learn and create models of normal application behavior. Results from our experiments with two online detection systems (stide and LERAD) indicate a drastic improvement in the number of attacks detected (without increasing the number of false alarms) during test phase when our technique is used for filtering the training set.

Merging argument and sequence information creates a richer model for anomaly detection. This relates to the issue of feature extraction and utilization for better behavioral modeling in a host-based system. We portrayed the efficacy of incorporating system call argument information and used a rule-learning algorithm to model a host-based anomaly detection system. Based upon experiments on well known data sets, we claim that our argument-based model, A-LERAD, detected more attacks than all the sequence-based techniques. Our sequence-based variant (S-LERAD) was also able to generalize better than the prevalent sequence based techniques, which rely on pure memorization.

The data cleaning procedure 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. 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 [37, 38, 39, 40]. We are also working on refining the motif relationships in the motif-based representation. Our argument and sequence based representations assume fixed size tuples.

A possible extension to variable length attribute window for more accurate modeling. Also, more sophisticated features can be devised from the argument information.

References

1. Denning, D.: An intrusion detection model. IEEE Transactions on Software Engineering **13** (1987) 222–232
2. Wagner, D., Soto, P.: Mimicry attacks on host-based intrusion detection systems. In: Computer and Communications Security. (2002)
3. Tan, K., Maxion, R.: Why 6? defining the operational limits of stide. In: IEEE Symposium on Security and Privacy, IEEE (2002) 188–201
4. Forrest, S., Hofmeyr, S., Somayaji, A., T., L.: A sense of self for unix processes. In: IEEE Symposium on Security and Privacy. (1996)
5. Lane, T., Brodley, C.: Detecting the abnormal: Machine learning in computer security. Technical Report TR-ECE 97-1, Purdue University, Lafayette, IN (1997)
6. Lane, T., Brodley, C.: Sequence matching and learning in anomaly detection for computer security. In: AI Approaches to Fraud Detection and Risk Management. (1997)
7. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Symposium on Security and Privacy. (1999)
8. Wespi, A., Dacier, M., Debar, H.: An intrusion-detection system based on the teiresias pattern-discovery algorithm. In: European Institute for Computer Anti Virus Research Conference. (1999)
9. Wespi, A., Dacier, M., Debar, H.: Intrusion detection using variable-length audit trail patterns. In: Recent Advances in Intrusion Detection. Lecture Notes in Computer Science. Springer-Verlag (2000)
10. Rigoutsos, I., Floratos, A.: Combinatorial pattern discovery in biological sequences. Bioinformatics (1998)
11. Ghosh, A., Schwartzbard, A.: A study in using neural networks for anomaly and misuse detection. In: USENIX Security Symposium. (1999)
12. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: Symposium on Security and Privacy. (2001)
13. Jiang, N., Hua, K., Sheu, S.: Considering both intra-pattern and inter-pattern anomalies in intrusion detection. In: International Conference on Data Mining. (2002)
14. Liao, Y., Vemuri, R.: Use of text categorization techniques for intrusion detection. In: USENIX Security Symposium. (2002)
15. Jones, A., Li, S.: Temporal signatures for intrusion detection. In: Annual Computer Security Applications Conference. (2001)
16. Coull, S., Branch, J., Szymanski, B., Breimer, E.: Intrusion detection: A bioinformatics approach. In: Annual Computer Security Applications Conference. (2003)
17. Mazeroff, G., De-Cerqueira, V., Gregor, J., Thomason, M.: Probabilistic trees and automata for application behavior modeling. In: ACM Southeast Regional Conference. (2003)

18. Lee, W., Stolfo, S., Chan, P.: Learning patterns from unix process execution traces for intrusion detection. In: AAAI workshop on AI methods in Fraud and risk management. (1997)
19. Cohen, W.: Fast effective rule induction. In: 12th International Conference on Machine Learning. (1995)
20. Portnoy, L.: Intrusion detection with unlabeled data using clustering. Technical report, Columbia University, NY (2000)
21. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., S., S.: A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In Barbara, D., Jajodia, S., eds.: Applications of Data Mining in Computer Security. Kluwer (2002)
22. Chan, P., Mahoney, M., Arshad, M.: Learning rules and clusters for anomaly detection in network traffic. In Kumar, V., Srivastava, J., Lazarevic, A., eds.: Managing Cyber Threats: Issues, Approaches and Challenges. Kluwer (2003)
23. Lazarevic, A., Ertoz, L., Ozgur, A., Srivastava, J., Kumar, V.: A comparative study of anomaly detection schemes in network intrusion detection. In: SIAM Conf. Data Mining. (2003)
24. Bernaschi, M., Gabrielli, E., Mancini, L.: Operating system enhancement to prevent the misuse of system calls. In: Computer and Communications Security. (2001)
25. Gibbs, A., McIntyre, G.: The diagram, a method for comparing sequences. its use with amino acid and nucleotide sequences. Eur. J. Biochem **16** (1970)
26. Mahalanobis, P.: On tests and measures of groups divergence. International Journal of the Asiatic Society of Bengal **26** (1930)
27. Knorr, E., Ng, R.: Algorithms for mining distance-based outliers in large data sets. In: VLDB, Morgan Kaufmann (1998)
28. Breunig, M., Kriegel, H., Ng, R., Sander, J.: Lof: Identifying density-based local outliers. In: SIGMOD. (2000) 93–104
29. Ramaswamy, S., Rastogi, R., Kyuseok, S.: Efficient algorithms for mining outliers from large data sets. In: ACM SIGMOD. (2000)
30. Aggarwal, C., Yu, P.: Outlier detection for high dimensional data. In: SIGMOD. (2001)
31. Salvador, S., Chan, P.: Learning states and rules for time series anomaly detection. In: FLAIRS, AAAI Press (2004)
32. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD. (1993) 207–216
33. Mahoney, M., Chan, P.: Learning rules for anomaly detection of hostile network traffic. In: International Conference on Data Mining. (2003)
34. Witten, I., Bell, T.: The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. In: Transactions on Information Theory. IEEE (1991)
35. Lippmann, R., Haines, J., Fried, D., Korba, J., Das, K.: The 1999 darpa off-line intrusion detection evaluation. Computer Networks (2000)
36. Whittaker, J., De-Vivanco, A.: Neutralizing windows-based malicious mobile code. In: ACM SAC, ACM (2002) 242–246
37. Weiner, P.: Linear pattern matching algorithms. In: 14th Annual IEEE Symposium on Switching and Automata Theory. (1973)
38. McCreight, E.: A space economical suffix tree construction algorithm. J. Assoc. Comput. Mach. **23** (1976) 262–272

39. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14** (1995) 249–260
40. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology. Cambridge University Press (1997)