Learning Rules from System Call Arguments and Sequences for Anomaly Detection

Gaurav Tandon and Philip Chan
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{gtandon, pkc}@cs.fit.edu

Abstract

Many approaches have been suggested and various systems been modeled to detect intrusions from anomalous behavior of system calls as a result of an attack. Though these techniques have been shown to be quite effective, a key element seems to be missing – the inclusion and utilization of the system call arguments to create a richer, more valuable signature and to use this information to model the intrusion detection system more accurately. We put forth the idea of adopting a rule learning approach that mobilizes rules based upon system calls and models the system for normal traffic using system call arguments and other key attributes. We present variations of our techniques and compare the results with those from some of the well known techniques based upon system call sequences. The results show that system call argument information is crucial and assists to successfully detect U2R, R2L and Data attacks generating lesser false alarms.

1. Introduction

Motivation: The Internet has invariably been a medium for malicious purposes. Attacks on computers, be it some graduate students trying to hack systems to prove their mettle or intruders with more damaging intentions, is on a steady rise. Moreover, novel attacks and hacking schemes are developed all the time, making it hard for systems to be made immune to all these vulnerabilities. It has thus become imperative that these be checked early to minimize losses.

Two different lines of approach have been adopted to detect intrusions. The first technique, *misuse (signature) detection*, is similar to pattern matching -- systems are modeled upon known attack patterns and the test data is checked for the occurrence of these patterns. These systems have a high degree of accuracy but fail to detect new attacks. The other method, *anomaly detection*, models normal behavior and significant deviations from this behavior are considered anomalous. The primary advantage of this approach is that it can detect novel attacks, the drawback being that it can generate a lot of false alarms. This is attributed to the fact that not all anomalies are necessarily attacks and will thus result in false positives.

Intrusion Detection Systems (IDSs) can also be categorized as network-based and host-based. In the former, header fields of the various network protocols are used to detect intrusions. For example, the IP header fields - source IP address, destination IP address, source port number, destination port number and others can be used to check for malicious intent. In the latter approach (a host-based IDS), the focus shifts to the operating system level. System call data is extracted from audit logs like the Solaris Basic Security Module (BSM) [16] and their behavior is studied to detect attacks.

Most of the present techniques for host-based anomaly detection systems revolve around sequences of system calls. These techniques are based upon the observation that an illegitimate activity results in an abnormal (novel) sequence of system calls.

Problem: The efficacy of such systems might be improved upon if more information is utilized. For system calls the most intuitive option lies in the system call arguments. Some other attributes related with system calls are the path for the object, the return value and the error status. Does adding these attributes assist in modeling a host-based anomaly detection system better? How do such systems fare (in terms of detections, false alarms, space and time

requirements) as compared to the systems based only upon the sequence of system call information? These are some of the key issues we seek to explore in this paper.

Approach: We extract system calls, their arguments, path, return value and error status from the Solaris BSM audit logs [16]. We then propose a host-based anomaly-detection system using system calls and other aforementioned key attributes by using variants of *LERAD* (Learning Rules for Anomaly Detection) [14], which is a conditional rule-learning algorithm. We aim at forming rules for our anomaly detection system based upon the system calls and their attributes. We suggest that including these attributes to the system calls will result in learning more information, thereby enabling us to model our systems better and detecting more attacks. We propose three models – the first one modeling system call sequences using *LERAD*, the second modeling system call arguments and other attributes, and the third approach being a combination of the two. We juxtapose these techniques and also compare them with some of the previous well-known sequence-based techniques, namely *tide*, *stide*, *t-stide* [20].

Contributions:

- We proposed the use of system call argument information to enrich the representation of program behavior in anomaly detection.
- We proposed modifications to *LERAD* to learn rules that allow one of the attributes to be designated as a pivotal attribute (system call in our case -- explanation in *Section 3.2.2*) on which the rules are based.
- As compared to *tide*, *stide* and *t-stide*, three well known sequence-based techniques (more details in *Section 2*), our argument-based systems are able to detect more attacks at lower false alarm rates.
- Our method that uses both sequence and argument information generally detected the most attacks with different false alarm rates.

Organization: *Section 2* describes the related work in the field of anomaly detection. In *Section 3*, we discuss the approach that we adopt for prepare the data set for our anomaly detection models. We give a brief explanation of *LERAD* on which our models are based. Then we describe the three variants of *LERAD* that are used to investigate different issues. *Section 4* gives a brief description of evaluation data, procedure and criteria. Then we analyze the results obtained from the experiments we performed. In *Section 5*, we conclude and put forth some views for future endeavors.

2. Related Work

Forrest et al. [2] proposed an approach for host based anomaly detection called time-delay embedding (*tide*), wherein traces of normal application executions were noted. A sliding look-ahead window of a fixed length was used to record correlations between pairs of system calls. These correlations were stored in a database of normal patterns, which was then used to monitor sequences during the testing phase. Anomalies were accumulated over the entire sequence and an alarm was raised if the anomaly count exceeded the threshold. *tide* forms correlations between pairs of system calls within a certain preset window size. Some of the issues involved in their approach were: using a small window does not help to form correlations over a long period of time. Similar sequences with minor variations could still be flagged as anomalous.

Later work by Warrender et al [20] extended this technique in sequence time-delay embedding (*stide*), which memorized all contiguous sequences of predetermined, fixed lengths during training. An anomaly count was defined as the number of mismatches in a temporally local region. A threshold was set for the anomaly score above which a sequence is flagged anomalous, indicating a possible attack. *stide* memorizes all fixed length sequences from the training data, irrespective of the number of instances found in the dataset. An extension, called sequence time-delay embedding with (frequency) threshold (*t-stide*), was similar to *stide* with the exception that the frequencies of these fixed length sequences were also taken into account. Rare sequences were ignored from the normal sequence database in this approach. When encountered during the testing phase, they were also counted as mismatches and aggregated to the locality frame counts (anomaly counts). All these techniques modeled normal behavior by using fixed length patterns of training sequences. But there was no rationale in fixing the length to a predetermined constant value.

Wespi et al. [21], [22] proposed a scheme to generate variable length patterns by using Teiresias [17], a pattern-discovery algorithm in biological sequences. These techniques improved upon the fixed length pattern methods cited above. Some extensions to (fixed and variable length) sequence-based methods were also proposed in [6], [7] and [8]. Though all the above mentioned approaches use system call sequences, none of them make use of the system call arguments. Given some knowledge about the system being used, attackers can devise some methodologies to evade such intrusion detection systems. Wagner and Soto [19] made such an

attempt to model a malicious sequence by adding "no-ops" (system calls having no effect) to compromise an IDS based upon the sequence of system calls. This brings to surface yet another shortcoming of sequence-based methods. Such attacks would fail if the system call arguments are also taken into consideration.

Sekar and others [18] proposed a method to build a compact finite state automaton (FSA) in an efficient way to detect intrusive activities. But no frequency information is stored in the FSA. Again, there lies the inherent drawback that the system call arguments are not considered. In [3], Feng et al proposed a method that dynamically extracts return address information from the call stack and program counter information is recorded at each system call. This technique performs equally well as compared to the deterministic FSA approach in terms of detections, convergence and false positives.

Artificial neural networks (ANNs) have been employed for both anomaly and misuse (signature) detection. Ghosh and Schwartzbad [4] expressed the idea of a process-based intrusion detection system that can generalize from previously observed behavior to recognize future unseen behavior. But their system ignores isolated anomalies.

Machine learning approaches have also been used to model intrusion detection systems. Lee at el. [11] verified the feasibility of rule-learning approaches by using an algorithm called *RIPPER* [1]. Mahoney and Chan [14] introduced a machine-learning algorithm called *LERAD* (Learning Rules for Anomaly Detection) to detect network intrusions. This technique extended the network traffic model to include a larger number of attributes. They also introduced and used the concept of a non-stationary model in [13], [14] and [15], in which the probability of an event depends upon its most recent occurrence and not on the frequency. *LERAD* is a conditional rule-learning algorithm that selects good rules from a vast rule space. This paper uses variants of *LERAD* for a host-based anomaly detection system.

3. Approach

Rule learning techniques have been shown that they can be successfully adapted to model systems for intrusion detection [14]. Since our goal is to detect host-based intrusions and we are dealing with BSM audit data, system calls are instrumental in our system. We thus extend upon the machine learning approach and incorporate the system calls with its arguments to generate a richer set of rules and measure the performance on the basis of number of detections and the false alarm rate. We study and evaluate three different variations of modeling a system using *LERAD*: sequence of system calls, system calls and their arguments, and a fusion of the previous two methodologies. We compare and contrast the results from these three models of our approach with *tide*, *stide* and *t-stide*.

3.1. Learning Rules for Anomaly Detection (*LERAD*)

LERAD is an efficient conditional rule-learning algorithm that picks up attributes in a random fashion. *LERAD* is briefly described here. More details can be obtained from [14]. *LERAD* learns rules of the form:

$$A = a, B = b, ... \Rightarrow X \in \{x_1, x_2, ...\} \quad (1)$$

where *A*, *B*, and *X* are attributes and *a*, *b*, *x1*, *x2* are values to the corresponding attributes. The learned rules represent the patterns present in the training data that consist of normal behavior. The set {*x1*, *x2*, …} in the consequent constitutes all unique values of *X* when the antecedent occurs in the training data. (These rules are different from typical classification rules or association rules.)

Records that match the antecedent but not the consequent of a rule are considered anomalous. The degree of anomaly is based on a probabilistic model. For each rule, from the training data, the probability, *p*, of observing a value not in the consequent is estimated by:

$$p = \Pr(X \notin \{x_1, x_2 ...\} \mid A = a, B = b, ...) = r / n \quad (2)$$

where '*r*' is the cardinality of the set, {x_1, x_2, …}, in the consequent and '*n*' is the number of records that satisfy the antecedent. This probability estimation of novel (zero frequency) events is due to Witten and Bell [23]. Since *p* estimates the probability of a novel event, the larger *p* is, the less anomalous a novel event is. Hence, during detection, when a novel event is observed, the degree of anomaly, or Anomaly Score, is estimated by:

$$AnomalyScore = 1/p = n/r \quad (3)$$

The rule generation phase of *LERAD* comprises of three main steps:

(i) Candidate rules are generated from patterns observed in randomly selected pairs of training examples: Training samples are picked up at random and then an initial set of rules is generated based upon common attributes between the samples. The conditional rules formed are of the type depicted in *Equation (1)* above.

(ii) The rule set is minimized by removing rules that do not cover/describe additional training examples: Redundant rules are discarded and a minimal set of rules is generated.

(iii) A subset of the training set is chosen as a validation set on which no training is performed: Rules learnt so far are used to test the data in this validation set. Rules are removed if they cause a false alarm in the validation set. This is due to the fact that the validation data set comprises of clean data (no attacks) and any anomaly implies a false alarm.

The rule generation methodology of *LERAD* is described next using *Table 1*.

Table 1: *LERAD* rule generation example: S1 – S6 are training samples with attributes A, B, C and D.

Training Sample	A	B	C	D
S1	1	2	3	4
S2	1	2	3	5
S3	6	7	8	4
S4	1	0	9	5
S5	1	2	3	4
S6	6	3	8	5

Step (i) Samples, say S1 and S2, are picked at random to create an initial rule set. Rules are generated by selecting matching attributes in a random order. In this example, the S1 and S2 have the matching attributes A, B and C. Selecting them in the order B, C and A, we get the following 3 rules:

Rule1: $* \Rightarrow B \in \{2\}$

Rule 2: $C=3 \Rightarrow B \in \{2\}$

Rule 3: $A=1, C=3 \Rightarrow B \in \{2\}$

A rule so generated implies that the attribute in the consequent can have a value from a set of values only if the conditions in the antecedent are satisfied. It may so happen that there is a consequent but no antecedent in a rule formed by *LERAD*. This means that an attribute can take any value from its set of values without the need to satisfy any other condition. Such a situation is presented in Rule 1 where the antecedent is represented by a wildcard character *.

Step (ii) Coverage test is applied to a subset of the training set (say S1-S3) and rules are modified as follows:

Rule1: $* \Rightarrow B \in \{2, 7\}$

Rule 2: $C=3 \Rightarrow B \in \{2\}$

Rule 3: $A=1, C=3 \Rightarrow B \in \{2\}$

Once we have the extended rule set, the probability p -- described in *Equation (2)* above -- is associated with every rule. The rules are then sorted in increasing order of the probability p:

Rule 2: $C=3 \Rightarrow B \notin \{2\}$ [$p = 1/2$]

Rule 3: $A=1, C=3 \Rightarrow B \notin \{2\}$ [$p = 1/2$]

Rule 1: $* \Rightarrow B \notin \{2, 7\}$ [$p = 2/3$]

When the probabilities are equal, the rule with lesser number of conditions in the antecedent is given higher priority (Rule 2 is higher in priority than Rule 3 in our example). Next, we desire a minimal set of rules. This is achieved by removing those rules that do not give any new information. In our example, Rule 2 is satisfied by samples 1 and 2. Rule 3 does not add any new value to the attribute B and is thus deemed as redundant and is removed from the rule set. The last rule (Rule 1) covers sample 3 as well and is kept in the rule set.

Extending the two rules to the entire training (minus validation) set (samples S1-S5 in our example), we get

Rule 2: $C=3 \Rightarrow B \notin \{2\}$ [$p = 1/3$]

Rule 1: $* \Rightarrow B \notin \{2, 7, 0\}$ [$p = 3/5$]

Step (iii): The last step comprises of testing the above set of rules on the validation set, which is a subset of the training data for which rules have not been generated. Any rule which produces anomaly in the validation set is removed. In our example, sample S6 forms the validation set. Rule 1 is violated since attribute B has a novel value 3 in this sample. Thus, we are left with the following rule:

C=3 $\Rightarrow$ B $\notin$ {2} [p = 1/3]

A non-stationary model is assumed for *LERAD* – frequency is made irrelevant and only the last occurrence of an event is assumed important. Since novel events are bursty in conjunction with attacks, a '*t*' factor was introduced to capture the non-stationary characteristic, where '*t*' is the time interval since the last novel (anomalous) event. When a novel event occurred recently, or *t* is small, a novel event is more likely to occur at the present moment. Hence, the anomaly score is measured by *t*/*p*. Since a record can deviate from the consequent of more than one rule, the total anomaly score of a record is:

$$TotalAnomalyScore = \sum_i t_i / p_i = \sum_i t_i n_i / r_i \qquad (4)$$

where '*i*' is the index of a rule from which the record has deviated. The anomaly score is aggregated over all the rules to combine the effect from violation of multiple rules. The more the violations, more critical the anomaly is, and the higher the anomaly score should be. *LERAD* yields successful results for network-based anomaly detection systems. This paper extends the algorithm for host-based anomaly detection systems.

3.2. Variants of *LERAD*

Our goal is to create a system that can detect any anomaly across any application/program. We developed a taxonomy of the entire data set from the BSM audit log. We classified the data into various applications/programs and generated a model for each of them.

3.2.1. Sequence of system calls: *S-LERAD*

Using sequence of system calls is a very popular approach for anomaly detection. We performed experiments wherein we extracted system calls from the data. We used a window of fixed length 6 (as this is claimed to give best results in *stide* and *t-stide* [20]) and fed these sequences of six system call tokens as input to *LERAD*. We called this technique as *S-LERAD* since we are trying to capture system call sequences by using *LERAD*.

For input to *LERAD*, we thus have a set of following attributes: date and time when system call information logged, the last two bytes of the destination IP address used for identifying the hosts during the evaluation, a system call and the previous five system calls, thereby making it a sequence of 6 system calls. *LERAD* uses these attributes at random to generate rules as described in *Section 3.1*.

The purpose of performing this experiment was to explore whether *LERAD* would be able to capture the correlations among system calls in a sequence. Also, this experiment would assist us in comparing results by using the same algorithm for system call sequences as well as system call arguments. Since *stide* and *t-stide* report best results for sequences of length 6, we increased the maximum number of allowed attributes in the antecedent of the rules generated by *LERAD* from 3 to 5, keeping the consequent fixed at 1 attribute.

A sample rule learned in a particular run of *S-LERAD* is:

$$SC1 = close(), SC2 = mmap(), SC6 = open() \Rightarrow SC3 \in \{munmap()\}$$

n/*r* value = 455/1

This rule is analogous to encountering *close*() as the first system call (represented as *SC 1*), followed by *mmap*() and *munmap*(), and *open*() as the sixth system call (*SC 6*) in a window of size 6 sliding across the audit trail. Each rule is associated with an n/*r* value, as explained in *Section 3.1*. The number 455 in the numerator refers to the number of training instances that comply with the rule (*n* in Equation 3). 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 Equation 3 of *Section 3.1*).

3.2.2. System call arguments and other key attributes: *A-LERAD*

We propose that argument and other key attribute information is integral to modeling a good host-based anomaly detection system. In this experiment, we extracted arguments, object path, return value and error status of system calls from the Solaris BSM audit log and examined the effects of learning rules based upon system calls along with these attributes.

We built models per application using *LERAD* with the modification that the rules were forced to have system call in the antecedent since it is the key attribute in a host based system. The generic version of *LERAD* could have been used to generate rules, but the motivation behind this is that ours is a host-based system and is centered upon system calls. We term the system call as a *pivotal attribute*

since our rules are based upon it. Thus, the system call will always be a condition in the antecedent of the rule.

This model is given the nomenclature *A-LERAD* since our motive here is to generate rules for various attributes given the system calls. Any value for the other arguments (given the system call) that was never encountered in the training period for a long time would raise an alarm. A sample rule is of the form:

$$SC = munmap() \Rightarrow Arg1 \in \{0x134, 0102, 0x211, 0x124\}$$
$$n/r \text{ value} = 500/4$$

In the above rule, 500/4 refers to the *n/r* value for the rule (*Equation 3* in *Section 3.1*), 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. This rule gives the 4 different values encountered for the first argument when the system call is munmap().

The maximum number of arguments has been chosen as 5 since most of system calls do not take more than 5 arguments. Considering more number of arguments results in more null values for the same and may cause formation of not-so-important rules thereby degrading the system performance. Thus only the high frequency arguments were selected from the data set. There may be several other approaches that can be adopted in this regard. Ours is just one intuitive approach.

3.2.3. Merging argument information and sequence of system calls: *M-LERAD*

The third set of experiments we conducted was to combine the techniques discussed in *Sections 3.3.1* and *3.3.2*. The first is a well acclaimed technique based upon sequence of system calls and is known to be an effective technique; the second one takes into consideration the attributes (arguments, path, return value and error status), whose efficacy we claim in this paper; so fusing the two to study the effects was an obvious choice. We call this technique as *M-LERAD* (short form for the merged system), as we desire to combine system call sequences and the related key attributes. Merging is accomplished by adding more attributes in each tuple before input to *LERAD*. Each tuple now comprises of the system call, arguments, object path, return value, error status and the previous five system calls. The *n/r* values obtained from the all rules violated are aggregated into an anomaly score, which is then used to generate an alarm based upon the threshold.

4. Experimental Evaluation

Our goal is to study if the rule-learning algorithm *LERAD* can be modified to determine as many attacks with least number of false alarms in a host-based anomaly detection system.

4.1. Evaluation Data and Procedures

We evaluated out techniques using the 1999 DARPA Intrusion Detection Evaluation Data Set [12]. The test bed involved a simulation of an air force base that has machines that are under frequent attack. These machines comprised of Linux, SunOS, Sun Solaris and Windows NT. Various intrusion detection systems were evaluated using this test bed, which comprised of three weeks of training data obtained from network sniffers, audit logs, nightly file system dumps and BSM logs from Solaris machine that trace system calls. Training was performed on week 3 data (around 2.1 million system calls) and testing on weeks 4 and 5 data (comprising over 7 million system calls) from the BSM audit log. A total of 51 attacks during weeks 4 and 5 were targeted at the Solaris machine, from which the BSM log was collected.

Data from the Basic Security Module (BSM) [16] audit log has to be preprocessed before it can be fed as input to *LERAD*. This was important from the point of view that we want to model process behavior for application. We divided the entire data set into various applications. For each application, we grouped the data on the basis of the process ID. For a given process id, all the data from the exec system call to the exit system call comprised the data for that particular process. Data for which we could not trace the start of the process was excluded from our experiments. The fork system call was dealt in a special way. A parent process spawns a child process with the fork system call, that is, a copy of the parent process is created. Unless fork is followed by exec, the child performs the same tasks as the parent process. Therefore, all the system calls for a child process are for the same application as the parent process until it encounters its own exec system call. In this way, we divided the data into applications, and further into processes belonging to the various applications/programs.

All the system calls (with their arguments) pertaining to a single process were thus differentiated from the set of

system calls (and arguments) for another process belonging to the same application. In a similar manner, sequences of system calls for various processes of different applications were differentiated from one another and were ready to be used for our rule-based learning models.

The parameters for *S-LERAD* were the 6 contiguous system calls; for *A-LERAD* they comprised of the system call, its return value and error status besides other arguments; and for *M-LERAD* it was a combination of the two techniques. For *tide*, the parameters were all the pairs of system calls within a window of fixed size 6; *stide* comprised all contiguous sequences of length 6, and *t-stide* added frequency information to the same. These sequence-based methodologies have been discussed in *Section 2*. In all models, alarms are accumulated for the applications and then evaluated for true detections and false positives.

4.2. Evaluation Criteria

The performance metrics used in this 1999 DARPA evaluation were the attack detection rate and the number of false alarms generated. We have adopted the same for the purpose of evaluating our system. As per the evaluation criteria, a system is considered to have successfully detected an attack if it generates an alarm within 60 seconds of the occurrence of the attack. We also follow the same criterion for evaluating our schemes.

The attacks in the 1999 DARPA evaluation are classified as probes, DoS, R2L, U2R and Data. These are based upon the classification by Kendell[10]. The taxonomy is as follows:

(i) Probes or scan attacks are attempts by hackers to collect information prior to an attack. Examples include illegalsniffer, ipsweep, mscan, portscan amongst others.

(ii) DoS (Denial of Service) attacks are the ones in which a host or a network service is disrupted. For example, arppoison, selfping, dosnuke and crashiis are all DoS attacks.

(iii) R2L (Remote to Local) – In these attacks, an unauthorized user gains access to a system. Examples of R2L attacks are guest, dict, ftpwrite, ppmacro, sshtrojan and framespoof.

(iv) U2R (User to Root) / Data attacks are those in which a local user is able to execute non-privileged commands, which only a super user can execute. Examples are eject, fdformat, ffbconfig, perl, ps and xterm.

Some attacks are combinations, such as a U2R attack that enables the attacker to steal secret data and are therefore categorized as Data-U2R attacks. Similarly, there are also Data-R2L attacks.

Lippmann et al [12] lists poorly detected attacks as the ones even half of whose instances were not detected by the any of the IDSs in the 1999 DARPA Evaluation. For the Solaris host, these were all DoS attacks. Host-based systems that use Solaris based audit data are more inclined to detect R2L, U2R and Data attacks than network-based intrusion detection systems.

As we are using more information (in the form of system call arguments) for our models, another important criterion is the space and the CPU time requirements, which is discussed in *Section 4.4*.

4.3. Results and Analysis of Detection Rates

We built training models for various applications. We reiterate our motivation for forcing rules based on system calls, as they are the pivotal attributes for our model. We trained our system on week 3 of the DARPA data and tested on weeks 4 and 5. Putative detections were considered as true positives if they occurred within 60 seconds of the attack segment for the correct destination (victim) IP address, which in our case was a single Solaris host.

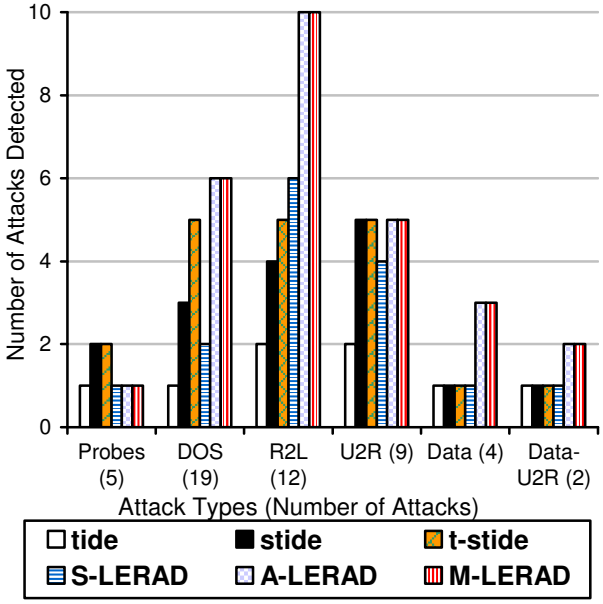

Figure 1: Number of detections with 10 false alarms per day for different attack categories.

Figure 1 plots the result based on a leeway of 10 false alarms per day of testing week, making a total of 100 false alarms for the two weeks of testing. The best technique using sequence-only information was *t-stide*, detecting 2 probes, 5 DoS, 5 R2L, 5 U2R, 1 Data and 1 Data-U2R attacks. Both *stide* and *t-stide* were able to find more probes than our argument-based technique, but our claim lies in finding more R2L, U2R and data attacks. *A-LERAD* was able to detect 10 R2L, 5 U2R, 3 Data, and both the Data-U2R attacks, apart from a probe and 6 DOS attacks. On the other hand, *S-LERAD* was not able to detect many of these attacks. The better performance of *A-LERAD* over *S-LERAD* can be attributed to the inclusion of argument information in the former model. The graph depicts no improvement by adding sequence information to argument information since *A-LERAD* and *M-LERAD* had exactly the same detections for the given false alarm rate. This also suggests that argument information is sufficient for detecting anomalies and there is no need for adding sequence information to *A-LERAD*.

Our techniques were also able to detect some poorly detected attacks quoted in [12]. For the Solaris host, these were *DoS* attacks, some of which we were able to capture accurately. There was only one instance of *tcpreset*, which our system detected successfully. We were also able to detect 2 instances of *warezclient*, both of which were not detected by the best system for that attack in the 1999 DARPA Evaluation.

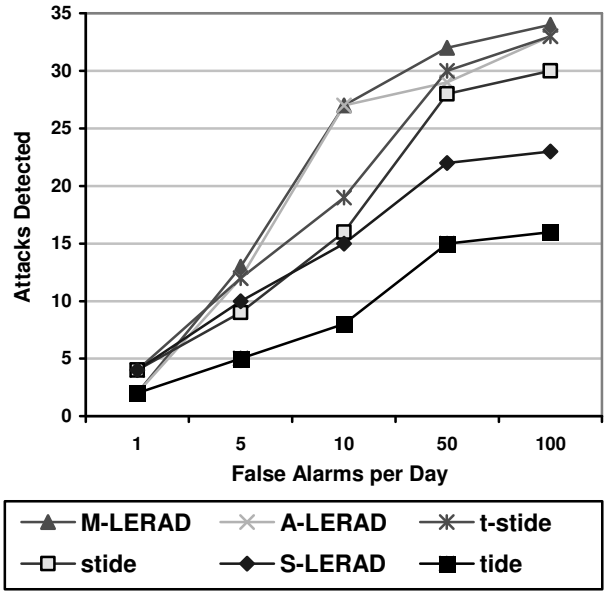

Figure 2: Detections for the 6 techniques at variable false alarms rates (for a total of 51 attacks in 2 weeks of data).

Figure 2 plots the total attacks detected by various techniques at 1, 5, 10, 50 and 100 false alarms per day respectively. *t-stide* maintained to be the best sequence-based technique, followed by *stide*, *S-LERAD* and *tide*. *A-LERAD* fared better than *S-LERAD* and the other sequence-based techniques, suggesting that argument information is more useful than sequence information. The *M-LERAD* curve is usually at or above the other curves, indicating that *M-LERAD* usually detects more attacks at various false alarm rates than the remaining five methods.

It can also be seen that the *A-LERAD* curve closely follows the curve for *M-LERAD*. This may imply that the sequence information is redundant; it is not adding substantial information to what we already have from the arguments. In other words, the attacks detected by using sequence information were also detected by using argument information, thereby giving similar results for *M-LERAD* and *A-LERAD*. A key point to observe is that even though the number of detections is almost same for the two techniques, *M-LERAD* has a faster convergence than *A-LERAD*.

We also observe that the significant difference in the performance of *M-LERAD* and *t-stide* is only at 10 false alarms per day. The reason for this is that the ROC curve is plotted on the basis of 5 discrete points only. For lower false alarm rates (1 and 5 per day), similar number of attacks was easily detected by both techniques. This can be attributed to the fact that these attacks contained both sequence and argument based anomalies. But as we increase the acceptable false alarm rate, we see that sequence anomalies do not necessarily correspond to an attack, whereas the argument anomalies are a good representation of an occurrence of an attack. By relaxing the allowed false alarm rate further (50 or 100 false alarms per day), we certainly expect to get more detections. We notice from the figure that we do get similar performance for *M-LERAD* and *t-stide* in such cases, but it is accompanied with a huge cost in terms of the number of false alarms, which is unacceptable for real-time systems.

By performing the comparison of the various techniques, we were also able to determine the effectiveness of the anomaly scoring function. Amongst the most effective techniques, *A-LERAD* and *M-LERAD* use a time based probabilistic estimation and *t-stide* incorporates frequency information. The way these techniques score anomalies is also a crucial factor in such anomaly detection systems.

One of the issues we investigated was whether to force *LERAD* to form rules based upon a system call as a condition in the antecedent or let it formulate rules without pivoting on a system call (as discussed in *Section 3.2.2*). We performed some experiments using *A-LERAD* with and without the enforcement of system call as a condition in the antecedent. Based upon the empirical evidence, we concluded that this enforcement resulted in the detection of at least as many attacks as in the relaxed case with the generation of fewer false alarms.

4.4. Results and Analysis of the CPU Time and Space Requirements

Compared to sequence-based methods, our techniques extract and utilize more information (system call arguments and other attributes), making it imperative to study the feasibility of our techniques in terms of space and time requirements.

During training, for *t-stide*, all contiguous system call sequences of length 6 along with their respective frequencies are stored in a database. For *M-LERAD*, system call sequences and other attributes are stored. In both the cases, space complexity is of the order of O(n), where 'n' is the total number of system calls, though the *M-LERAD* requirement is more by a constant factor k since it stores additional argument information. During detection, *M-LERAD* uses only the learned set of rules (in the range 14-35 at an average of 25.1 rules per application in our experiments). *t-stide*, on the other hand, still requires the entire database of fixed length sequences during testing, which incur larger space overhead during detection. We conducted experiments on the *tcsh* application data. The entire week 3 training data set comprises of over 2 million system calls and the test data (weeks 4 and 5 combined) has over 7 million system calls. For *tcsh,* system calls alongwith their arguments form a 33 MB input file for *M-LERAD*. The rules formed by *M-LERAD* require less than 1.5 KB space, apart from a mapping table to map strings and integers. For the same application, the memory requirements for storing a system call sequence database for *t-stide* were over 5 KB plus a mapping table between strings and integers. The results suggest that *M-LERAD* has better memory requirements during the detection phase. We reiterate that the training can be done offline. Once the rules are generated, *M-LERAD* can be used to do online testing with lower memory requirements.

The time overhead incurred by *M-LERAD* and *t-stide* in our experiments is given in *Table 2*. The CPU times have been obtained on a Sun Ultra 5 workstation with 256 MB RAM and 400 MHz processor speed. We can infer from the results that *M-LERAD* is slower than *t-stide*. During training, *t-stide* is a much simpler algorithm and processes less data than *M-LERAD* for building a model and hence *t-stide* has a much shorter training time. During detection, *t-stide* just needs to check if a sequence in the database, which can be efficiently implemented with a hash table. On the other hand, *M-LERAD* needs to check if a record matches any of the learned rules. Also, *M-LERAD* has to process additional argument information. Run-time performance of *M-LERAD* can be improved with more efficient rule matching algorithm. Also, *t-stide* will incur significantly larger time overhead when the stored sequences exceed the memory capacity and disk accesses become unavoidable – *M-LERAD* does not encounter this problem as easily as *t-stide* since it will still use a small set of rules. More importantly, *M-LERAD*'s time overhead is about tens of seconds for days of data, which is reasonable for practical purposes.

Table 2: Comparison of CPU times during training and testing phases for *t-stide* and *M-LERAD* for top 8 applications in terms of total number of system calls (not necessarily in that order).

Application	Training Time (seconds) [on 1 week of data]		Testing Time (seconds) [on 2 weeks of data]	
	t-stide	M-LERAD	t-stide	M-LERAD
ftpd	0.19	0.99	0.19	0.96
telnetd	0.96	7.87	1.05	9.79
ufsdump	6.76	33.33	0.42	1.78
tcsh	6.32	32.85	5.91	37.58
login	2.41	16.75	2.45	19.86
sendmail	2.73	15.09	3.23	21.63
quota	0.20	3.48	0.20	3.79
sh	0.21	3.25	0.40	5.63

5. Concluding Remarks

Even though system call sequences are beneficial in modeling normal process behavior, they are not omniscient. In this paper, we portrayed the efficacy of incorporating system call argument information and used a rule-learning algorithm to model a host-based anomaly detection system. Our argument-based model, *A-LERAD*, detected more attacks at lower false alarm rates than the sequence-based techniques on the 1999 DARPA evaluation dataset. Combining the two lines of approach

(argument and sequence information) resulted in creating a richer and, more importantly, more accurate model for anomaly detection, as illustrated by the empirical results of *M-LERAD*. Though our techniques incur higher time overhead due to the complexity of our techniques as well as more information to be processed, they build more succinct models that incur much less space overhead--our techniques aim to generalize from the training data, rather than simply memorize the data.

Our techniques can be easily extended to monitor audit trails in continuum. Since we model each application separately, some degree of parallelism can also be achieved to test process sequences as they are being logged. *S-LERAD* fares poorly as compared to *stide* and *t-stide*. We are currently trying to analyze and rectify its shortcomings, which might have an impact on the performance of *M-LERAD* as well. Also, we were able to see from our experiments that the time based probabilistic estimation of anomaly score as proposed in *LERAD* and the frequency component of *t-stide* are effective ways to flag data as anomalous. These two functions can be combined to give a more appropriate anomaly score. It would be interesting to see how this would affect the results. We might perform experiments and publish results for the same in the near future.

Acknowledgements

This work is partially funded by DARPA (F30602-00-1-0603). We thank the LLR members for their help on ideas and the anonymous reviewers for their comments.

References

[1] Cohen W. Fast Effective Rule Induction, in Machine Learning. *Proc. ICML* 1995.

[2] Forrest S., Hofmeyr S., Somayaji A., and Longstaff T. A Sense of Self for UNIX Processes. *1996 IEEE Symposium on Research in Security and Privacy*.

[3] Feng H., Kolesnikov O., Fogla P., Lee W. and Gong W. Anomaly Detection Using Call Stack Information. *IEEE Symposium on Security and Privacy,* 2003.

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

[5] Hangal S. and Lam M.S. Tracking Down Software Bugs Using Automatic Anomaly Detection. *International Conference on Software Engineering,* 2002.

[6] Helman P. and Bhangoo J. A statistically based system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 1997.

[7] Hofmeyr S. A., Forrest S., and Somayaji A. Intrusion detection using sequences of system calls. *Journal of Computer Securit,y* 1998.

[8] Jiang N., Hua K., and Sheu S. Considering Both Intra-pattern and Inter-pattern Anomalies in Intrusion Detection. *Proc. Intl. Conf. Data Mining*, 2002.

[9] Jones A., Li S. Temporal Signatures for Intrusion Detection. *Computer Security Applications Conference*, 2001.

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

[11] Lee W., Stolfo S., and Chan P. Learning Patterns from UNIX Process Execution Traces for Intrusion Detection. *AAAI'97 workshop on AI methods in Fraud and risk management*.

[12] Lippmann R., Haines J., Fried D., Korba J., and Das K. The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks*, 2000.

[13] Mahoney M. and Chan P. Packet Header Anomaly Detection for Identifying Hostile Network Traffic, *Florida Tech. Technical Report CS-2001-04*.

[14] Mahoney M., and Chan P. Learning Rules for Anomaly Detection of Hostile Network Traffic, *Proc. of the Third IEEE International Conference on Data Mining, 2003 (to appear)*.

[15] Mahoney M. and Chan P. Learning non-stationary models of normal network traffic for detecting novel attacks. *Proc. Intl. Conf. Knowledge Discovery and Data Mining,* P 376-385, 2002.

[16] Osser W., and Noordergraaf A. *Auditing in the Solaris™ 8 Operating Environment*. Sun Blueprints™ Online - February 2001.

[17] Rigoutsos Isidore and Floratos Aris. Combinatorial pattern discovery in biological sequences. *Bioinformatics*, 1998.

[18] Sekar R., Bendre M., Dhurjati D., Bollineni P. A Fast Automaton-based Method for Detecting Anomalous Program Behaviors. *IEEE Symposium on Security and Privacy*, 2001.

[19] Wagner D., Soto P. Mimicry Attacks on Host-Based Intrusion Detection Systems. *ACM Conference on Computer and Communications Security*, 2002.

[20] Warrender C., Forrest S., Pearlmutter B. Detecting Intrusions Using System Calls: Alternative Data Models. *IEEE Symposium on Security and Privacy*, 1999.

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

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

[23] Witten I. and Bell T., The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. on Information Theory*, 1991.