International Journal on Artificial Intelligence Tools

ON THE LEARNING OF SYSTEM CALL ATTRIBUTES FOR HOST-BASED ANOMALY DETECTION

GAURAV TANDON
Department of Computer Sciences, Florida Institute of Technology, 150 W. University Blvd. Melbourne, Florida 32901, USA
gtandon@cs.fit.edu

PHILIP K. CHAN
Department of Computer Sciences, Florida Institute of Technology, 150 W. University Blvd. Melbourne, Florida 32901, USA
pkc@cs.fit.edu

Traditional host-based anomaly detection systems model normal behavior of applications by analyzing system call sequences. The current sequence is then examined (using the model) for anomalous behavior, which could correspond to attacks. Though these techniques have been shown to be quite effective, a key element is missing – the inclusion and utilization of the system call arguments. Recent research shows that sequence-based systems are prone to evasion. We propose an idea of learning different representations for system call arguments. Results indicate that this information can be effectively used for detecting more attacks than traditional sequence-based techniques, with reasonable storage and computational overhead.

Keywords: Anomaly detection; host-based intrusion detection; system call arguments.

1. Introduction

Intrusion detection systems (IDSs) are generally categorized as signature-based and anomaly-based. In signature detection, systems are modeled upon known attack patterns and the test data is checked for the occurrence of these patterns. Such systems have a high degree of accuracy but suffer from the inability to detect novel attacks. Anomaly detection complements signature detection by modeling normal behavior of applications. Significant deviations from this behavior are considered anomalous. Such systems can detect novel attacks, but generate false alarms since not all anomalies are necessarily hostile. Intrusion detection systems can also be categorized as network-based, which deals with network traffic; and host-based, where operating system events are monitored.

Most of the traditional host-based anomaly detection systems focus on system call sequences, the assumption being that a malicious activity results in an abnormal (novel) sequence of system calls. Recent research has shown that sequence-based systems can be compromised by conducting mimicry attacks. Such attacks are pos-

sible by inserting dummy system calls with invalid arguments such that they form a legitimate sequence of events.

A drawback of sequence-based approaches lies in their non-utilization of other key attributes, namely 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 paper we present a host-based anomaly detection system that is based upon system call arguments. We learn the important attributes using a variant of a rule learning algorithm called LERAD. We also present various argument-based representations and compare their performance with some of the well-known sequence-based techniques.

Our main contributions are:

- we propose enriched representations using various system call attributes (return value, error status and other arguments) for better application modeling;
- we use a variant of a rule learning algorithm to learn the important attributes from the feature space;
- we demonstrate the effectiveness of our models (in terms of number of attack detections, time and space overhead) by performing experiments on three different data sets; and
- we present an analysis of the anomalies detected.

Our argument-based models are able to detect more attacks than their sequence-based counterparts. The time and storage requirements for our models are reasonable for online detection.

2. Related Work

Time-delay embedding (tide) records executions of normal application executions using look-ahead pairs.[1] A fixed length sliding window records correlations between pairs of system calls. Anomalies are accumulated over the entire sequence and an alarm is raised if the anomaly count exceeded the threshold. UNIX command sequences were also examined to capture user profiles and compute sequence similarity using adjacent events in a sliding window.[2] Sequence time-delay embedding (stide) memorizes all contiguous sequences of predetermined, fixed lengths during training.[3] An anomaly count is defined as the number of mismatches in a temporally local region. A further 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. All these techniques modeled normal behavior by using fixed length patterns of training sequences. A scheme to generate variable length patterns by using Teiresias,[4] a pattern-discovery algorithm in biological sequences, was proposed.[5,6] Though all the above approaches use system call sequences, none of them make use of the system call arguments. Given some knowledge about the IDS, attackers can devise some methodologies to

evade such intrusion detection systems.[7,8] Such attacks might be detected if the system call arguments are also evaluated,[9] and this motivates our current work. Our technique models only the important characteristics and generalizes from them; previous work emphasizes on the structure of all the arguments.

3. Approach

Since our goal is to detect host-based intrusions, system calls are instrumental in our system. We incorporate the system calls with its arguments to generate a richer model. Then we present different representations for modeling a system using LERAD, which is discussed next.

3.1. *Learning Rules for Anomaly Detection*

Algorithms for finding association rules, such as Apriori,[10] generate **all** rules that exceed a user defined support and confidence. A large number of rules could incur substantial overhead for rule checking and might not be appropriate for on-line detection. We would like to have a minimal set of rules describing the normal training data. Moreover, association rules do not allow a set of values in the consequent. LERAD[11] is a conditional rule-learning algorithm that forms a small set of rules. LERAD learns rules of the form:

$$A = a \wedge B = b \wedge \ldots \Rightarrow X \in \{x_1, x_2, \ldots\} \tag{1}$$

where A, B, and X are attributes and a, b, x_1, x_2 are values for the corresponding attributes. The learned rules represent the patterns present in the normal training data. The set $\{x_1,\ x_2,\ \ldots\}$ in the consequent constitutes all unique values of X when the antecedent occurs in the training data.

During the detection phase, records (or tuples) that match the antecedent but not the consequent of a rule are considered anomalous and an anomaly score is associated with every rule violation. The degree of abnormality is based on the probability estimation of novel (zero frequency) events.[12] For each rule, from the training data, the probability, p, of observing a value not in the consequent is estimated by:

$$p = \frac{r}{n} \tag{2}$$

where r is the cardinality of the set, $\{x_1,\ x_2,\ \ldots\}$, in the consequent and n is the number of records (tuples) that satisfy the rule during training. p is the probability of observing a novel event ($P(Event = novel)$), not the probability of observing each event ($P(Event = value)$). That is, we are not interested in the relative probability of observed events in the training data, rather, we are interested in events **not** observed in the training data and have **zero** frequency. In a sequence of observations, a novel event is one that has not been observed previously in the sequence. The rate of seeing novel events is the number of unique events (r) divided by the number

of observations (n). This rate is used to estimate p, the probability of observing a novel event.

Consider two sequences of events: 1110001100 and 1020506079, where an event is represented by an integer in the closed interval [0,9]. It can be seen that the first sequence has a lower degree of randomness than the second sequence. Hence the probability of a novel subsequent event for the former sequence (equal to 0.2) is lower than the latter sequence, where it is 0.7, using Eq. 2. 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 abnormality (anomaly score) is estimated by:

$$AnomalyScore = \frac{1}{p} = \frac{n}{r} \tag{3}$$

A non-stationary model is assumed for LERAD only the last occurrence of an event is assumed important. Since novel events are bursty in conjunction with attacks, a factor t is introduced which is the time interval since the last novel (anomalous) attribute value. If a novel event occurred recently (small value of t), a novel event is more likely to occur at the present moment. Conside two sequences of events: 1111000010 and 1111100000. Even though both sequences have two distinct events, the first sequence is more likely (and hence less alarming) to encounter a novel susbequent event since it has seen change more recently than the second sequence. Thus, the anomaly score is measured by t/p. For simplicity, no non-linearities are considered in the anomaly score.

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:

$$TotalAnomalyScore = \sum \frac{t}{p} = \sum \frac{tn}{r} \tag{4}$$

The more the violations, more significant the anomaly is, and the higher the anomaly score should be. An alarm is raised if the total anomaly score is above a threshold.

The rule generation phase of LERAD comprises four main steps:

(i) Generate initial rule set: Training samples are picked up at random from a random subset S of training examples. Candidate rules (as depicted in Eq. 1) are generated from these samples.
(ii) Coverage test: The rule set is filtered by removing rules that do not cover/describe all the training examples in S. Rules with lower rate of abnormalities (lower r/n) are kept.
(iii) Update rule set beyond S: Extend the rules over the remaining training data by adding values for the attribute in the consequent when the antecedent is true.
(iv) Validate the rule set: Rules are removed if they are violated by any tuple in the validation set.

Since system call is the key (pivotal) attribute in a host based system, we modified LERAD such that the rules were forced to have a system call as a condition in the antecedent. The only exception we made was the generation of rules with no antecedent.

3.2. *System call and argument based representations*

We conjecture that even though augmentation of attributes to system call sequences may tend to make the hypotheses space more complex, it will also enable an algorithm to learn the hypotheses more accurately. We now present the different representations for LERAD.

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

Using sequence of system calls is a very popular approach for anomaly detection. We used a window of fixed length six (as this is claimed to give best results in stide and t-stide [1,3]) and fed these sequences of six system call tokens as input tuples to LERAD. This representation is selected to explore whether LERAD would be able to capture the correlations among system calls in a sequence. Also, this experiment would assist us to compare and better evaluate argument based models. A sample rule learned in a particular run of S-LERAD is:

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

This rule is analogous to encountering *close* as the first 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 six 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. 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 Eq. 3).

3.2.2. *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 extracted arguments, return value and error status of system calls from the audit logs and examined the effects of learning rules based upon system calls along with these attributes. 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 typical argument based rule is

$$(s_0 = close) \Rightarrow a_0 \in \{0x2, 0x3, 0x4, 0x5, 0x6\}$$
$$[1/\mathrm{p} = n/r = 500/5]$$

where a_0 is the first argument for the first system call (*close*).

We performed experiments on the training data to measure the maximum number of attributes (MAX) for every unique system call. We did not use the test data for these experiments so that we do not get any information about it before our model is built. Since LERAD accepts the same (fixed) number of attributes for every tuple, we had to insert a NULL value for an attribute that was absent in a particular system call. The order of the attributes within the tuple was made system call dependent. By including all attributes we utilized the maximum amount of information possible.

It can be argued that inclusion of NULL values in a rule may result in the formation of many not-so-important rules, thereby making the rule set large and incurring high time and space overhead. But it is also important to note that even though NULL values do not seem to be adding any useful information, they could still help to detect anomalies. Consider the rule

$$(s_0 = munmap) \wedge (a_0 = 0x10) \Rightarrow a_2 \in \{NULL\}$$
$$[1/\mathrm{p} = n/r = 2000/1]$$

The rule has only one acceptable value for the third argument when the system call (s_0) is *munmap* and the first argument is 0x10. The rule has a large coverage (= 2000 tuples) in the training data (assuming 2000 tuples is large for a given system call with respect to entire data set). Thus, one would never expect to see any other value for the third argument when the antecedent is true. However, an intruder could craft a mimicry attack by introducing arbitrary argument values (without realizing what a valid value for that argument should be), resulting in a novel value. In such a situation we could say with high confidence that it is a highly unexpected event and hence depicts anomalous behavior. Even though the NULL value for the argument might not determine the nature of the attack, it could be decisive in detecting the anomaly introduced due to a slight carelessness on the part of the intruder.

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

The first representation we discussed is based upon sequence of system calls; the second one takes into consideration other relevant attributes, whose efficacy we claim in this paper; so fusing the two to study the effects was an obvious choice. Merging is accomplished by adding more attributes in each tuple before input to LERAD. Each tuple now comprises the system call, MAX number of 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 an anomaly score, which is then used to generate an alarm based upon the threshold. A sample rule for M-LERAD is of the form

$$(s_0 = close) \wedge (s_5 = open) \Rightarrow a_0 \in \{0x4, 0x5\}$$

$$[1/p = n/r = 107/2]$$

where s_0 and s_5 correspond to the system calls at the extremities of the sliding window and a_0 is the first argument for the current system call (i.e. *close*).

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

A-LERAD and M-LERAD take system call arguments into account for the current system call. We extend this representation further in 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 all system calls within the fixed length sequence of size six. Each tuple now comprises the current system call, MAX attributes for the current system call, 5 previous system calls and MAX attributes for each of those system calls. An actual rule formed by M*-LERAD is

$$(s_0 = close) \wedge (a_{5,1} = 4) \wedge (s_4 = mmap) \Rightarrow a_{3,3} \in \{0xe000\}$$
$$[1/p = n/r = 117/1]$$

In the rule above, s_0 and s_4 are the first and fifth system calls respectively in the current window, $a_{5,1}$ is the second argument for the last system call and $a_{3,3}$ is the fourth argument for the fourth system call in the sliding window.

4. Experimental Evaluation

The goal is to study if our variant of LERAD with feature spaces comprising system calls and their arguments can detect attack-based anomalies.

4.1. *Data sets and experimental procedure*

We used the following data sets for our experiments:

(i) The 1999 DARPA intrusion detection evaluation data set: Developed at the MIT Lincoln Lab,[13] we selected the Basic Security Module (BSM) logs from Solaris host tracing system calls that contains 33 attacks. The attack taxonomy[14] is briefly explained here.
   (a) Probes or scan attacks are attempts by hackers to collect information prior to an attack. Examples include illegalsniffer, ipsweep, mscan, portscan amongst others.
   (b) 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.
   (c) R2L (Remote to Local) are those attacks wherein an unauthorized user gains access to a system. Examples of R2L attacks are guest, dict, ftpwrite, ppmacro, sshtrojan and framespoof.

(d) 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.

The following applications were chosen: *ftpd*, *telnetd*, *sendmail*, *tcsh*, *login*, *ps*, *eject*, *fdformat*, *sh*, *quota* and *ufsdump*, due to their varied sizes (about 1500 to over one million system calls). We expected to find a good mix of benign and malicious behavior in these applications. Training was performed on week 3 data and testing on weeks 4 and 5. An attack is considered to be detected if an alarm is raised within 60 seconds of its occurrence (same as the DARPA evaluation).

(ii) *lpr*, *login* and *ps* applications from the University of New Mexico (UNM): The *lpr* application comprised of 2703 normal traces from hosts running SUNOS 4.1.4. Another 1001 traces result from the execution of the *lprcp* attack script. Traces from the *login* and *ps* applications were obtained from Linux machines running kernel 2.0.35. Trojan programs were used for the attack traces.

(iii) Microsoft *excel* macros executions (FIT-UTK data): Normal *excel* macro executions are logged in 36 distinct traces. Two malicious traces modify registry settings and execute some other application. Such a behavior is exhibited by the ILOVEYOU worm which opens the web browser to a specified website and executes a program, modifying registry keys and corrupting user files, resulting in a distributed denial of service (DDoS) attack.

The input tuples for S-LERAD were six contiguous system calls; for A-LERAD they were system calls with their return value, error status and arguments; The inputs for M-LERAD were sequences of system calls with arguments of the current system call; whereas in M*-LERAD, they were system call sequences with arguments for all the six system calls. For tide, the inputs were all the pairs of system calls within a window of fixed size six; stide and t-stide comprised all contiguous sequences of length six. For all the techniques, alarms were merged in decreasing order of the anomaly scores and evaluated at varied false alarm rates.

4.2. *Comparison of sequence based methods*

t-stide is supposed to give best results among the traditional sequence-based techniques.[3] As the UNM and FIT-UTK data sets do not have complete argument information, we only evaluated and compared S-LERAD and t-stide on these data sets. The receiver operating characteristic (ROC) curves from this set of experiments are displayed in Fig. 1. The X-axis in the plots corresponds to various false alarm rates and the Y-axis represents the percentage of attacks detected. The drawback of anomaly detection lies in the generation of false alarms. Typically, a human expert (administrator) has to deal with all the alarms and the cost of wrongly flagging an alarm can be quite high. Our goal is thus to detect as many attacks as possible while minimizing the generation of false alarms. We have therefore limited

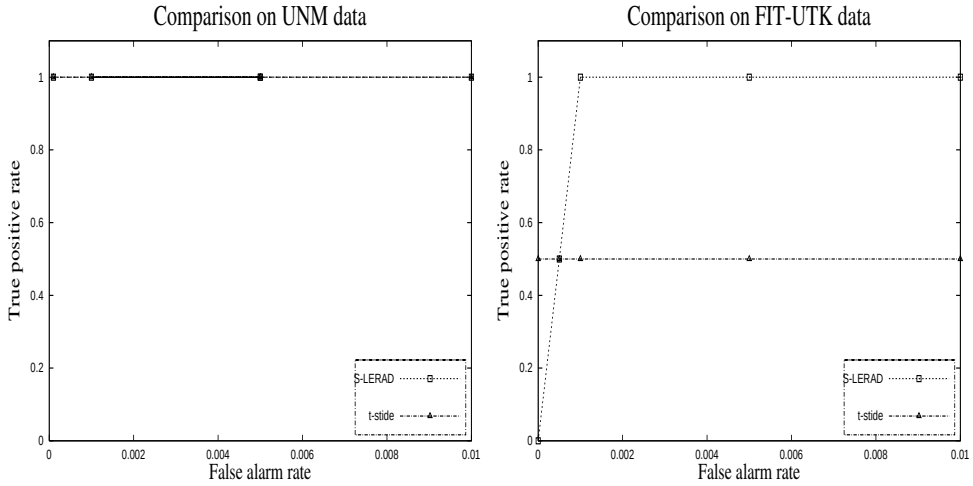

Fig. 1. ROCs for UNM And FIT-UTK data.

our ROCs to a maximum of 1% false alarms (0.01 on the X-axis). Results from Fig. 1 show that both the techniques were able to detect all the attacks for the UNM data without generating any false alarm. However, for the FIT-UTK data set, there was a difference in the performance. t-stide was initially able to detect more attacks, but S-LERAD was able to detect all the attacks at 0.1% false alarm rate.

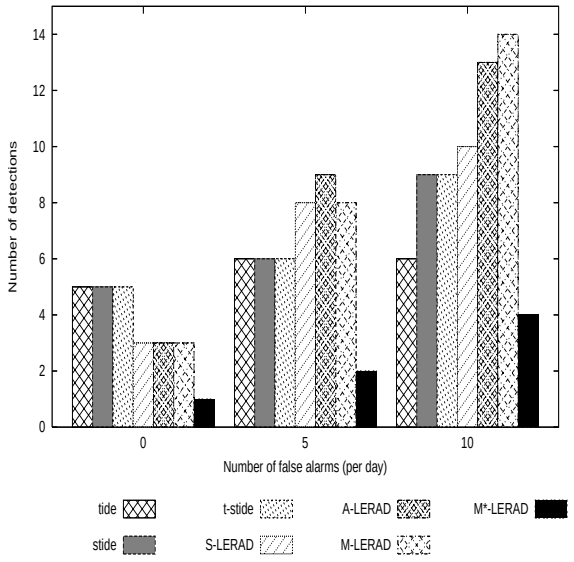

Fig. 2. Number of detections at different false alarm rates for the BSM data.

We also performed experiments on the DARPA BSM data sets to evaluate all the techniques. Fig. 2 illustrates the total attacks detected (Y-axis) at different false alarms rates (X-axis). The original DARPA evaluation[13] used a threshold of 10 false alarms per day. We used the same evaluation criterion for consistency. The tolerance limit of allowed false alarms may vary from one organization to the other. A government organization dealing with sensitive data would not want to miss any anomaly, keeping the threshold high. On the other hand, a small company may not have the infrastructure to sustain the cost associated with many false alarms per day, thereby preferring a lower threshold but at the risk of missing some intrusions. In order to encompass the viability for a broader range of organizations, we also evaluated the techniques at 0 and 5 false alarms per day. At zero false alarms, tide, stide and t-stide detected the most attacks, suggesting that maximum deviations in temporal sequences are true representations of actual attacks. But as the threshold is relaxed, S-LERAD outperformed all the 3 sequence-based techniques. This can be attributed to the fact that S-LERAD is able to generalize well and learns the important correlations between the system calls within the given window size.

For completeness, we also present the ROC curves for various techniques in Fig. 3. As can be observed from the figure, S-LERAD performs better at false alarm rates less than 0.1%. But t-stide detects more attacks at higher false alarm rates.

We also list the area under the curve (up to 1% false alarms) in Table 1, where higher area implies better performance.[15] The area under curve of the rectangle with false alarm rate of range [0,0.01] and detection rate of range [0,1] is 0.01. We consider a detector with an area under its ROC curve of 0.01 as the *perfect* detector (with at most 0.01 false alarm rate). The detector that decides randomly has a diagonal line from (0,0) to (0.01, 0.01), called the uninformative ROC, and its area is $0.01 \times 0.01/2$ or 0.00005, which is 0.5% of the perfect detector. Results in the table show that overall t-stide performs better than S-LERAD in the range of 0 to 1% false alarm rates.

There are two noteworthy points about the DARPA evaluation criterion that we used in our experiments. Firstly, the evaluation had a strict requirement of 10 false alarms per day (Fig. 2), which corresponds to less than 0.1% false alarms on the ROC curve (Fig. 3). Secondly, an attack is considered detected only if an alarm is raised within one minute of its occurrence. Thus, an alarm after 60 seconds would be considered a false positive even though it may have resulted due to some unusual activity by the intruder.

4.3. *Comparison of system call sequence and argument based methods*

For the DARPA BSM data set, A-LERAD fared better than S-LERAD and the other sequence-based techniques when evaluated in terms of both absolute number (Fig. 2) and the percentage (Fig. 3) of false alarms, suggesting that argument infor-

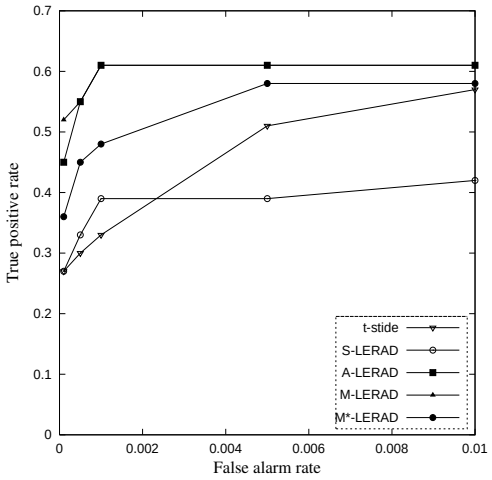

Fig. 3. ROC for BSM data.

Table 1. Area under curve up to 1% false alarm rates. Equivalent area of the uninformative prediction (diagonal) is 0.5%.

Technique	Area under curve (%)
t-stide	46.79
S-LERAD	39.12
A-LERAD	60.25
M-LERAD	60.46
M*-LERAD	54.51

mation is more useful than sequence information. Using arguments could also make a system robust against mimicry attacks which evade sequence-based systems. It can also be seen from Fig. 3 that the A-LERAD curve closely follows the curve for M-LERAD. This implies 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, as can be seen in Fig. 2. 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 (Fig. 3). Table 1 also shows that A-LERAD and M-LERAD perform the best under 1% false alarm rates, followed by M*-LERAD and S-LERAD.

Generally, the better performance of LERAD 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 abnormality. 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;
(ii) suppresses the generation of multiple alarms for novel attribute values in a sudden burst of data.

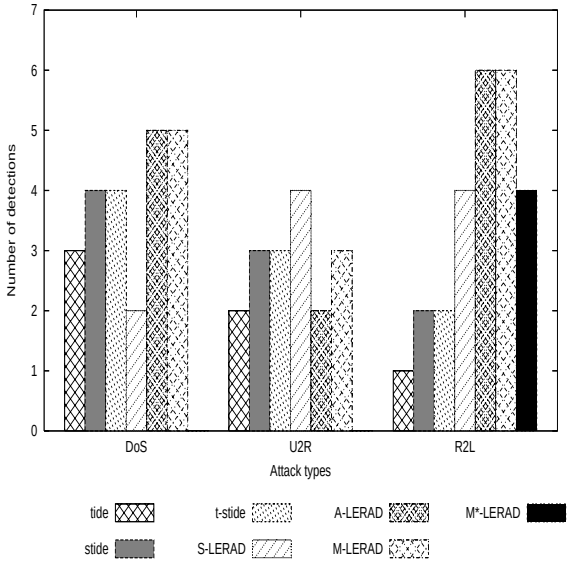

Fig. 4. Types of attacks detected at 10 false alarms per day for the BSM data.

Figure 4 plots the different types of attacks (mentioned in Sec. 4.1) detected at 10 false alarms per day (criterion used in the 1999 DARPA evaluation). Different attack types (DoS, U2R and R2L) are represented along the X-axis and the Y-axis denoted the total attacks detected in each attack category. M-LERAD was able to detect the largest number of attacks 5 DoS, 3 U2R and 6 R2L attacks A-LERAD followed closely with 5 DoS, 2 U2R and 6 R2L attacks. 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 techniques were able to detect some poorly detected attacks,[13] *warezclient* being one of them. Our models also detected stealthy *ps* attacks.

4.4. *NULL attribute values*

As described in Section 3.2.2, we inserted NULL values for attributes not present for a given system call. The expectation is that LERAD would form a rule if the NULL attribute value is characteristic for a system call in an application. Experiments were performed to see if NULL attributes help in detecting anomalies or if they formed

meaningless rules. We added a constraint that the NULL values could not be added to the attribute values in the rules and called this feature variant A^C-LERAD (A-LERAD with constraint). Table 2 compares the number of attacks detected by A-LERAD and A^C-LERAD by varying the false alarm rate. Results indicate that A-LERAD was able to detect more attacks than the constrained counterpart, suggesting that rules with NULL valued attributes are beneficial to the detection of anomalies corresponding to attacks.

Table 2. Attacks detected by A-LERAD and A^C-LERAD.

False Alarms per day	Number of attacks detected	
	A-LERAD	A^C-LERAD
5	10	9
10	13	11
20	17	16

To analyse why A-LERAD was able to perform better, two metrics were devised to characterize the rule set:

(i) Rule Ratio (RR): the ratio of the number of rules formed by A^C-LERAD to the number of rules formed by A-LERAD. This ratio conveys which technique formed a smaller rule set, resulting in a lower run-time overhead.
(ii) Coverage Ratio (CR): Coverage is the count of the tuples in the training set that are covered by the rule set. Since the coverage is descriptive of the number of tuples it describes, we would want a high value for coverage. Coverage ratio is the ratio of the coverage of A^C-LERAD to the coverage of A-LERAD.

Table 3. Comparison of rule ratio and coverage ratio.

Application	Rule Ratio (RR)	Coverage Ratio (CR)
ftpd	0.35	0.51
telnetd	0.33	0.33
tcsh	0.44	0.48
sendmail	0.52	0.56
quota	0.50	0.53
login	0.36	0.38
sh	0.29	0.27
eject	0.62	0.56
ps	0.50	0.55
ufsdump	0.50	0.49

RR should ideally be low since a small rule set is desired. The coverage ratio should be close to 1, indicating that a comparable number of tuples are covered by A^C-LERAD and A-LERAD. Table 3 shows the results for various applications in the

BSM log. For example, the size of the rule set for A^C-LERAD was 35% of the size of the rule set of A-LERAD for *ftpd*. This implies fewer rules for A^C-LERAD and hence a lower run-time overhead during the detection phase. However, the coverage ratio was 51%. The anomaly score is higher for rules with higher coverage (Eq. 3). This assists A-LERAD to assign higher scores to tuples that violate rules and hence detect more attacks than A^C-LERAD. Overall results indicate that even though discarding NULL attribute values results in a drastic reduction in the rule set, they also cause smaller coverage amongst the training tuples. Another interesting observation from Table 3 was that a decrease in the size of the rule set generally yields a similar decrease in the coverage.

4.5. *Analysis of anomalies - attack detections and false alarms*

An anomaly is a deviation from normalcy and, by definition, does not necessarily identify the nature of an attack. Anomaly detection serves as an early warning system; humans need to investigate if an anomaly actually corresponds to a malicious activity. Table 4 list the anomalous attributes that resulted in the detection of the attack. It is interesting to note that the argument based anomalies that led to the attacks detected by LERAD, in many cases, do not represent the true nature of the attacks. Instead, it may be representative of behavioral patterns resulting from the execution of some other program after the intruder successfully gained access to the host. For example, an instance of *guest* attack is detected by A-LERAD not by observing attempts by the hacker trying to gain access, but by encountering novel arguments to the *ioctl* system call which was executed by the hacker trying to perform a control function on a particular device. A stealthy *ps* attack was detected by our system when the intruder tried to change owner using a novel group id.

Table 4. Anomalous arguments for attacks detected by A-LERAD.

Attack	Attribute anomalies
ftpwrite	set audit without ownership, invalid file descriptor, invalid mmap return value, invalid device-dependent request code
guesstelnet	novel ioctl return value
ps	change group ownership with anomalous group id, invalid device, inappropriate ioctl for device
guessftp	fail file open on remote system
warez	invalid device-dependent request code, change ownership of file without privileges
guest	inappropriate ioctl for device
warezclient	not owner to set audit state
syslogd	novel ioctl argument

Even if the anomaly is related to the attack itself, it may reflect very little information about the attack. Our system is able to learn only a partial signature of the attack. *guessftp* is detected by a bad password for an illegitimate user trying to gain access. However, the attacker could have made interspersed attempts to evade the system. Attacks were also detected by capturing errors committed by the intruder, possibly to evade the IDS. *ftpwrite* is a vulnerability that exploits a configuration error wherein a remote ftp user is able to successfully create and add files and gain access to the system. An instance of this attack is detected by monitoring the subsequent actions of the intruder, wherein he attempts to set the audit state using an invalid preselection mask. This anomaly would go unnoticed in a system monitoring only system calls.

We re-emphasize that our goal is to detect anomalies, the underlying assumption being that anomalies generally correspond to attacks. Since not all anomalous events are malicious, we expect false alarms to be generated. Table 5 lists the attributes responsible for the generation of alarms and whether these resulted in actual detections or not. It is observed that some anomalies were part of benign application behavior. At other instances, the anomalous value for the same attribute was responsible for detecting actual malicious execution of processes. As an example, many attacks were detected by observing novel arguments for the *ioctl* system call, but many false alarms were also generated by this attribute. Even though not all novel values correspond to any illegitimate activity, argument-based anomalies were instrumental in detecting the attacks.

Table 5. Top anomalous attributes for A-LERAD.

Attribute causing false alarm	Whether some attack was detected by the same attribute
ioctl argument	Yes
ioctl return value	Yes
setegid mask	Yes
open return value	No
open error status	No
fcntl error status	No
setpgrp return value	No

4.6. *Storage and computational overheads*

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 for online usage. For t-stide, all contiguous system call sequences of length six are stored during training. For A-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

A-LERAD requirement is more by a constant factor k since it stores additional argument information.

During detection, A-LERAD uses only a small set of rules (in the range 10-25 for the applications used 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 *tcsh* application in the BSM data set, which comprises over 2 million system calls in training and has over 7 million system calls in test data. The rules formed by A-LERAD require around 1 KB space, apart from a mapping table to map strings and integers. 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 A-LERAD has better memory requirements during the detection phase. We reiterate that the training can be done offline. Once the rules are generated, A-LERAD can be used to do online testing with lower memory requirements.

Table 6. Computational overhead for one week training and two weeks testing.

Application	Total training time (seconds)		Testing time per sample (milliseconds)	
	tstide	ALERAD	tstide	ALERAD
ftpd	0.19	0.90	0.04	0.16
telnetd	0.96	7.12	0.02	0.17
tcsh	6.32	29.56	0.13	0.17
sendmail	2.73	14.79	0.05	0.17
quota	0.20	3.04	0.01	0.14
login	2.41	15.12	0.04	0.17
sh	0.21	2.98	0.03	0.16
ufsdump	6.76	30.04	0.01	0.14

The time overhead incurred by A-LERAD and t-stide in our experiments is given in Table 6. The CPU times have been obtained on a Sun Ultra 5 workstation with 256 MB RAM and 400 MHz processor speed. It can be inferred from the results that A-LERAD is slower than t-stide. During training, t-stide is a much simpler algorithm and processes less data than A-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 is present in the database, which can be efficiently implemented with a hash table. On the other hand, A-LERAD needs to check if a record matches any of the learned rules. Also, A-LERAD has to process additional argument information. Run-time performance of A-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 A-LERAD does not encounter this problem as easily as t-stide since it will still use a small set of rules. Moreover, the run-time overhead of A-LERAD is below a couple of hundred microseconds per sample, which is reasonable for practical purposes.

5. Conclusions

Sequence based anomaly detection systems are known to suffer from high false alarm rates. 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. We performed experiments on data sets from varied operating systems and applications. We empirically demonstrated that the argument-based model, A-LERAD, detected more attacks than all the sequence-based techniques, stressing on the importance of system call arguments for modeling host based systems.

Merging argument and sequence information creates a richer model for anomaly detection, as illustrated by the empirical results of M-LERAD. M*-LERAD detected lesser number of attacks at lower false alarm rates since every anomalous attribute results in alarms being raised in six successive tuples, leading to either multiple detections of the same attack (counted as a single detection) or multiple false alarms (all separate entities). Results also indicated that sequence-based methods help detect U2R attacks whereas R2L and DoS attacks were better detected by argument-based models. The argument-based models detected different types of anomalies. Some anomalies did not represent the true nature of the attack. Some attacks were detected by subsequent anomalous user behavior, like attempts to change group ownership. Some other anomalies were detected by learning only a portion of the attack, while some were detected by capturing intruder errors.

Though our proposed variants incur higher time overhead (due to the complexity of LERAD and enriched feature space) as compared to t-stide, they build more succinct models that incur much less space overhead our techniques aim to generalize from the training data, rather than pure memorization. Moreover, under 200 microseconds per sample (during testing phase) is reasonable for online systems, even though it is significantly longer than t-stide.

Though our representations detected more attacks with fewer false alarms, there arises a need for more sophisticated attributes. Instead of having a fixed sequence, we could extend our models to incorporate variable length sub-sequences of system calls. Even the argument-based models are of fixed window size, creating a need for a model accepting varied argument information. 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. The rule learning algorithm can also be modified further by associating penalty with rules violated during the validation phase, instead of eliminating them completely.

References

1. S. Forrest, S. Hofmeyr, A. Somayaji and T. Longstaff, A Sense of Self for UNIX Processes. *IEEE Symposium on Security and Privacy*, (1996), pp.120–128.
2. T. Lane and C.E. Brodley, Sequence Matching and Learning in Anomaly Detection

for Computer Security. *AAAI workshop on AI Approaches to Fraud Detection and Risk Management*, (1997).
3. C. Warrender, S. Forrest and B. Pearlmutter, Detecting Intrusions Using System Calls: Alternative Data Models. *IEEE Symposium on Security and Privacy*, (1999), pp.133–145.
4. I. Rigoutsos and A. Floratos, Combinatorial pattern discovery in biological sequences. *Bioinformatics*, 14(1), (1998), pp.55–67.
5. A. Wespi, M. Dacier and H. Debar, An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. *European Institute for Computer Anti-Virus Research*, (1999), pp.1–15.
6. A. Wespi, M. Dacier and H. Debar, Intrusion detection using variable-length audit trail patterns. *Recent Advances in Intrusion Detection*, (2000), pp.110–129.
7. K.M.C. Tan, K.S. Killourhy and R.A. Maxion, Undermining an Anomaly-based Intrusion Detection System Using Common Exploits. *Recent Advances in Intrusion Detection*, (2002), pp.54–73.
8. D. Wagner and P. Soto, Mimicry Attacks on Host-Based Intrusion Detection Systems. *ACM Conference on Computer and Communications Security*, (2002), pp.255–264.
9. C. Kruegel, D. Mutz, F. Valeur and G. Vigna, On the Detection of Anomalous System Call Arguments, *European Symposium on Research in Computer Security*, (2003), pp.326–343.
10. R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases. *ACM SIGMOD*, (1993), pp.207–216.
11. M. Mahoney and P. Chan, Learning Rules for Anomaly Detection of Hostile Network Traffic, *IEEE International Conference on Data Mining*, (2003), pp.601–604.
12. I. Witten and T. Bell, The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, vol.37, (1991), pp.1085–1094.
13. R. Lippmann, J. Haines, D. Fried, J. Korba and K. Das, The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks*, 34(4), (2000), pp.579–595.
14. K. Kendell, A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Masters Thesis, MIT, (1999).
15. P.A. Flach, The Many Faces of ROC Analysis in Machine Learning. *International Conference on Machine Learning Tutorial*, (2004).