Learning Useful System Call Attributes for Anomaly Detection

Gaurav Tandon and Philip K. Chan

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

Abstract

Traditional host-based anomaly detection systems model normal behavior of applications by analyzing system call sequences. 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 seems to be 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 with reasonable space and time overhead.

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 possible 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 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 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: (1) we incorporate various system call attributes (return value, error status and other arguments) for better application modeling; (2) we propose enriched representations using system call sequences and arguments; (3) we use a variant of a rule learning algorithm to learn the important attributes from the feature space; (4) 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 (5) we present an analysis of the anomalies detected. Our sequence-based model detects more attacks than traditional techniques, indicating that the rule learning technique is able to generalize well. Our argument-based systems are able to detect more attacks than their sequence-based counterparts. The time and space requirements for our models are reasonable for online detection.

Related Work

Time-delay embedding (tide) records executions of normal application executions using look-ahead pairs (Forrest et al. 1996). UNIX command sequences were also examined to capture user profiles and compute sequence similarity using adjacent events in a sliding window (Lane and Brodley 1997). Sequence time-delay embedding (stide) memorizes all contiguous sequences of predetermined, fixed lengths during training (Warrender, Forrest, and Pearlmutter 1999). 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 (Rigoutsos and Floratos 1998), a pattern-discovery algorithm in biological sequences, was proposed in (Wespi, Dacier, and Debar 1999, 2000). These techniques improved upon the fixed length methods. 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 (Tan, Killourhy, and Maxion 2002; Wagner and Soto 2002). Such attacks might be detected if the system call arguments are also evaluated (Kruegel et al. 2003), and this motivates our current work. Our technique models only the important characteristics and generalizes from it; previous work emphasizes on the structure of all the arguments.

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.

Learning Rules for Anomaly Detection (LERAD)

Algorithms for finding association rules, such as Apriori (Agrawal, Imielinski, and Swami 1993), generate a large number of rules. This incurs a large overhead and may not be appropriate for online detection. We would like to have a *minimal* set of rules describing the normal training data. LERAD is a conditional rule-learning algorithm that forms a small set of rules. It is briefly described here; more details can be obtained from (Mahoney and Chan 2003). LERAD learns rules of the form:

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

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 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:

$$p = r / n \qquad (2)$$

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. This probability estimation of novel (zero frequency) events is from (Witten and Bell 1991). 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 (anomaly score) is estimated by:

$$\textit{Anomaly Score} = 1 / p = n / r \qquad (3)$$

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 – it is the time interval since the last novel (anomalous) attribute value. When a novel event occurred recently (small value of t), 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 aggregated over all the rules violated by the tuple to combine the effect from violation of multiple rules:

$$\textit{Total Anomaly Score} = \sum t / p = \sum t\, n / r \qquad (4)$$

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 of 4 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 Equation 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 anomalies (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.

System call and argument based representations

We now present the different representations for LERAD.

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 6 (as this is claimed to give best results in stide and t-stide) 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 in comparing results by using the same algorithm for system call sequences as well as their arguments. A sample rule learned in a particular run of *S-LERAD* is:

$$SC_1 = close, SC_2 = mmap, SC_6 = open \Rightarrow SC_3 \in \{munmap\}$$

(1/*p* 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. 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).

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.

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. Since we modified LERAD to form rules based upon the system calls, there is consistency amongst the attributes for any specific system across all models. By including all attributes we utilized the maximum amount of information possible.

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 of 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.

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 all system calls within the fixed length sequence of size 6. Each tuple now comprises of the current system call, *MAX* attributes for the current system call, 5 previous system calls and *MAX* attributes for each of those system calls.

Experimental Evaluation

Our goal is to study if LERAD can be modified to detect attack-based anomalies with feature spaces comprising system calls and their arguments.

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, we selected the BSM logs from Solaris host tracing system calls that contains 33 attacks. Attack classification is provided in (Kendell 1999). The following applications were chosen: *ftpd*, *telnetd*, *sendmail*, *tcsh*, *login*, *ps*, *eject*, *fdformat*, *sh*, *quota* and *ufsdump*, due to their varied sizes (1500 – over 1 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 collected from 77 hosts running SUNOS 4.1.4 at the MIT AI Lab. 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. Homegrown 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. 2 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 6 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 6 system calls. For tide, the inputs were all the pairs of system calls within a window of fixed size 6; stide and t-stide comprised all contiguous sequences of length 6. For all the techniques, alarms were merged in decreasing order of the anomaly scores and evaluated at varied false alarm rates.

Results

Since t-stide is supposed to give best results among the sequence-based techniques, we compared its performance with S-LERAD on the UNM and FIT-UTK data sets.

Table 1: t-stide vs. S-LERAD (UNM, FIT-UTK data).

Program name	Number of training sequences	Number of test sequences	Number of attacks detected (Number of false alarms)	
			t-stide	S-LERAD
lpr	1000	2704	1 (0)	1 (1)
ps	12	27	2 (58)	2 (2)
login	8	8	1 (0)	1 (1)
excel	32	6	2 (92)	2 (0)

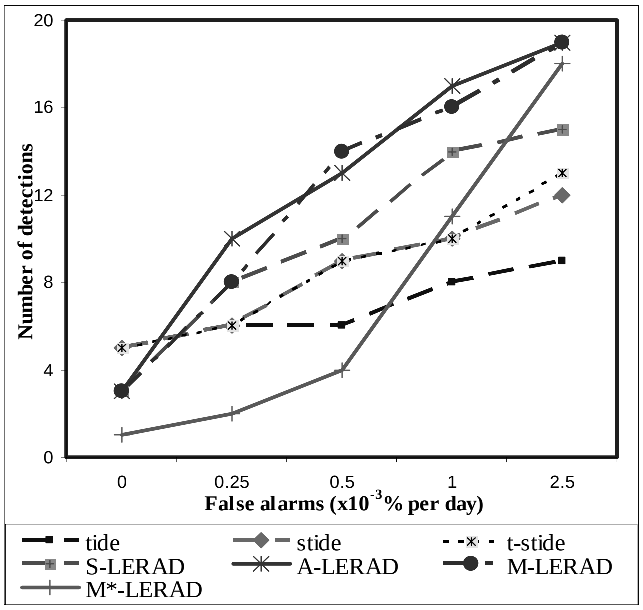

Figure 1. Number of detections (DARPA/LL data).

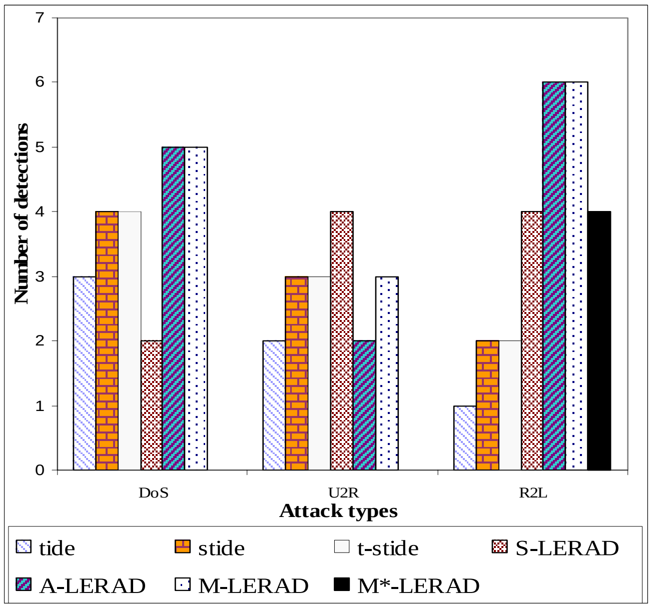

Figure 2. Number of detections at 10 false alarms per day for different attack categories (DARPA/LL data).

Results from Table 1 show that both the techniques were able to detect all the attacks. However, t-stide generated more false alarms for ps and excel. We also performed experiments on the DARPA/LL data sets to evaluate all the techniques. Figure 1 illustrates the total attacks detected (Y-axis) at varied false alarms rates (X- axis). 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.

The UNM and FIT-UTK data sets do not have complete argument information to evaluate LERAD variants that involve arguments. For the DARPA/LL data set, A-LERAD fared better than S-LERAD and the other sequence-based techniques (Figure 1), suggesting that argument information 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 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 at false alarms rate lower than 0.5 x 10^{-3} % per day. 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.

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; (ii) suppresses the generation of multiple alarms for novel attribute values in a sudden burst of data.

Figure 2 plots the result at 10 false alarms per day, making a total of 100 false alarms for the 10 days of testing (criterion used in the 1999 DARPA evaluation). Different attack types 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. 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 quoted in (Lippmann et al. 1999), *warezclient* being one of them. Our models also detected 3 stealthy *ps* attacks.

Table 2. A-LERAD vs. A^{C}-LERAD (DARPA/LL).

False alarms per day	Number of detections	
	A-LERAD	A^{C}-LERAD
5	10	9
10	13	11
20	17	16

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. We call this variant A^C-LERAD (A-LERAD with constraint). Table 2 summarizes the results. 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.

Analysis of anomalies

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. The anomalies that led to the attacks detected by argument-based variants of 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.

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 (such as .rhost) and gain access to the system. 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.

Table 3. 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

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 3 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.

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 for online usage. For t-stide, all contiguous system call sequences of length 6 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 14-25 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 the *tcsh* application, which comprises of 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.

The time overhead incurred by A-LERAD and t-stide in our experiments is given in Table 4. 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 about tens of seconds for days of data, which is reasonable for practical purposes.

Table 4. Execution time comparison.

Application	**Training Time (seconds) [on 1 week of data]**		**Testing Time (seconds) [on 2 weeks of data]**	
	t-stide	**A-LERAD**	**t-stide**	**A-LERAD**
ftpd	0.19	0.90	0.19	0.89
telnetd	0.96	7.12	1.05	9.79
ufsdump	6.76	30.04	0.42	1.66
tcsh	6.32	29.56	5.91	29.38
login	2.41	15.12	2.45	15.97
sendmail	2.73	14.79	3.23	19.63
quota	0.20	3.04	0.20	3.01
sh	0.21	2.98	0.40	3.93

Conclusions

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. Based upon experiments on various 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.

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 6 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. Our argument-based techniques 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 trying 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 techniques incur higher time overhead due to the complexity of our techniques (since more information is processed) 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, 3 seconds per day (the most an application took during testing phase) is reasonable for online systems, even though it is significantly longer than t-stide.

Though our techniques did detect 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.

References

Agrawal, R.; Imielinski, T.; and Swami A. 1993. Mining association rules between sets of items in large databases. *ACM SIGMOD*, 207-216.

Forrest, S.; Hofmeyr, S.; Somayaji, A.; and Longstaff, T. 1996. A Sense of Self for UNIX Processes. *IEEE Symposium on Security and Privacy*, 120-128.

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

Kruegel, C.; Mutz, D.; Valeur, F.; and Vigna, G. 2003. On the Detection of Anomalous System Call Arguments, *European Symposium on Research in Computer Security*, 326-343.

Lane, T., and Brodley C.E. 1997. Sequence Matching and Learning in Anomaly Detection for Computer Security. *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 43-49.

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

Mahoney, M., and Chan, P. 2003. Learning Rules for Anomaly Detection of Hostile Network Traffic, *IEEE International Conference on Data Mining*, 601-604.

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

Tan, K.M.C.; Killourhy, K.S.; and Maxion, R.A. 2002. Undermining an Anomaly-based Intrusion Detection System Using Common Exploits. *RAID*, 54-74.

Wagner, D., and Soto, P. 2002. Mimicry Attacks on Host-Based Intrusion Detection Systems. *ACM CCS*, 255-264.

Warrender, C.; Forrest, S.; and Pearlmutter, B. 1999. Detecting Intrusions Using System Calls: Alternative Data Models. *IEEE Symposium on Security and Privacy*, 133-145.

Wespi, A.; Dacier, M.; and Debar, H. 1999. An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. *EICAR Conference*, 1-15.

Wespi, A.; Dacier, M.; and Debar, H. 2000. Intrusion detection using variable-length audit trail patterns. *RAID*, 110-129.

Witten, I., and Bell, T. 1991. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. on Information Theory*, 37(4):1085-1094.