Proceedings of the 2005 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY

Learning Classifiers for Misuse and Anomaly Detection Using a Bag of System Calls Representation

Dae-Ki Kang, Doug Fuller, Vasant Honavar

Abstract— **In this paper, we propose a "bag of system calls" representation for intrusion detection in system call sequences and describe misuse and anomaly detection results with standard machine learning techniques on University of New Mexico (UNM) and MIT Lincoln Lab (MIT LL) system call sequences with the proposed representation. With the feature representation as input, we compare the performance of several machine learning techniques for misuse detection and show experimental results on anomaly detection. The results show that standard machine learning and clustering techniques on simple "bag of system calls" representation of system call sequences is effective and often performs better than those approaches that use foreign contiguous subsequences in detecting intrusive behaviors of compromised processes.**

I. Introduction

Detection of attempts to compromise the integrity, confidentiality, or availability of computing and communication networks is an extremely challenging problem [1]. Most current approaches to the design of intrusion detection systems (IDS) are based on the premise that the actions used in an attempted intrusion can be differentiated from the actions executed by users or processes during the normal operation of the computing and communication networks [2]. An effective IDS logs actions executed by users or processes for investigation, alerts the system administrator when the monitored activities are indicative of attempted intrusion, and, if appropriate, takes corrective measures e.g., expelling the intruder.

Intrusion detection and prevention generally refers to a broad range of strategies for defending against malicious attacks. Intrusion detection can be categorized into *misuse* detection and *anomaly* detection. Misuse typically is a known attack, e.g., a hacker attempting to break into an email server in a way that IDS has already trained. A misuse detection system tries to model normal and abnormal behavior from known attacks. It works by comparing network traffic, system call sequences, or other features of known attack patterns. An anomaly is something out of the ordinary, e.g., abnormal network traffic which is actually caused by unknown attacks. An anomaly detection system models *normal* behavior and identifies a behavior as abnormal (or anomalous) if it is sufficiently different from known normal behaviors.

Supported by NSF grant IIS 0219699.
Department of Computer Science, Iowa State University, Ames, IA.

IDS can be classified into those that focus on modeling the behavior of users and those that focus on modeling the behavior of processes [3]. System call data are one of the most common types of data used to model the behavior of processes. Such data can be collected by logging the system calls using operating system utilities e.g. Linux strace or Solaris Basic Security Module (BSM).

There has been a great deal of research on how to design and implement intrusion detection systems. For example, Mukherjee et al [4] used a combination of host monitors and network monitors with a centralized director for suspicious system activities in the distributed intrusion detection system (DIDS) project. Because it is difficult to manually specify activities that signal intrusive behavior, there has been much work on adaptive or machine learning or data mining approaches for intrusion detection. Forrest et al [5] worked on the Computer Immunology project and explored approaches inspired by the activities of the immune systems of animals for detecting and defending against intrusions. Subsequently, several groups have explored data mining approaches for intrusion detection [6], [7], [8].

In most IDS that model the behavior of processes, intrusions are detected by observing fixed-length, contiguous subsequences of system calls. For example, in anomaly detection, subsequences of input traces are matched against normal sequences in database so that foreign sequences are detected. [5], [9] One potential drawback of this approach is that the size of the database that contains fixed-length contiguous subsequences increases exponentially with the length of the subsequences. For example, if the number of system calls is 200 and the length of the subsequences is 6, the maximum size of the database is theoretically $200^6 = 64 \times 10^{12}$ words. In practice, only normal subsequences are stored, so actual database size is smaller, but still considerably bigger than the hypothesis size generated by the approach presented here.

In this paper, we explore an alternative representation of system call traces for intrusion detection. We use a *bag of system calls* representation of system call sequences and consider intrusion detection of system call sequence as a classification problem on a bag of system calls obtained from the system call sequences. With those prob-

0-7803-9290-6/05/$20.00 ©2005 IEEE.

lem setting, we constructed and evaluated decision tree learner [10], Naive Bayes learner [11], rule learner [12], support vector machines (SVM) [13], [14], and logistic regression model (with a ridge estimator) [15] using bag of system calls representation of system calls. We also explored application of the representation to anomaly detection tasks using a one class Naive Bayes classifier as well as K-means clustering [16] using the same representation of system call sequences. Bag of words model is already popular in text classification and categorization area [17], and our motivation is to investigate the usefulness of the model in intrusion detection tasks.

The results show that the proposed approach for misuse detection yields comparable or sometimes better performance than the methods previously reported in the literature in terms of detection rate and false positive over widely used benchmark data sets such as University of New Mexico (UNM) and MIT Lincoln Lab (MIT LL) system call sequences.

The rest of the paper is organized as follows. Section 2 describes two different representations of system call sequences. Section 3 describes the benchmark data sets used in our study. Section 4 describes the experimental setup and results. Section 5 concludes with a summary and discussion.

II. Alternative Representations of System Call Sequences

We describe two feature representations of system call sequences that intrusion detection algorithms deal with. The first one is a contiguous subsequence with fixed length k from original input traces, and the second is bag of system calls, which is our approach.

One of the main questions in sequence-based intrusion detection is how to define "intrusion" in an input sequence. Most intrusion detection algorithms such as *STIDE* [18] regard that intrusion is related with fixed-length subsequence that only happens in intrusive traces.

In our approach, we convert the input sequence to bag of system calls. Thus, the ordering information between system calls is lost and only the frequency of each system call is preserved for each input sequence. Intrusion is represented according to the machine learning algorithm applied.

Formally, the intrusion detection problem on system call or command sequences can be defined as follows:

Let $\Sigma = \{s_1, s_2, s_3, \ldots, s_m\}$ be a set of system calls where $m = |\Sigma|$ is the number of system calls. Data set D can be defined as a set of labeled sequences $\{< Z_i, c_i > | Z_i \in \Sigma^*, c_i \in \{0, 1\}\}$ where Z_i is an input sequence and c_i is a corresponding class label denoting 0 for "normal" label and 1 for "intrusion" label. Given the data set D, the goal of the learning algorithm is to find a classifier $h : \Sigma^* \rightarrow \{0, 1\}$ that maximizes given criteria. Widely accepted criteria include accuracy, detection rate and false positive rate.

Because it is difficult to deal with sequences directly, each sequence $Z \in \Sigma^*$ is mapped into a finite dimensional feature vector by a feature representation $\Phi : \Sigma^* \rightarrow \mathbf{X}$. Thus, the classifier is defined as $h : \mathbf{X} \rightarrow \{\mathbf{0}, \mathbf{1}\}$ for data set $\{< X_j, c_j > | X \in \mathbf{X}, \mathbf{c_j} \in \{\mathbf{0}, \mathbf{1}\}\}$.

A. Contiguous Foreign Subsequences

In this approach, a feature is defined as $X_j = x_1 x_2 x_3 \ldots x_k$, a substring of Z_i, where $x_{1 \leq l \leq k} \in \Sigma$ and k is a constant. The maximum number of distinct features is $|\Sigma|^l$ and each feature X_j is assigned a class label c_i according to the original sequence Z_i.

STIDE uses sliding windows with length k over an original input trace to generate fixed-length substrings as features and constructs a database of the features in the training stage, and decides a test sequence is anomalous if the number of mismatches in the user-specified locality frame (locality frame count), which is composed of adjacent features in the frame, is more than the user-specified threshold. Empirically, it is widely accepted that, for effective intrusion detection, the minimal value of k is six [19].

B. Bag of System Calls

"Bag of system calls" representation is an integer-frequency based method. In our approach, a feature is defined as an ordered list $X_i = \langle c_1, c_2, c_3, \ldots, c_m \rangle$ where $m = |\Sigma|$ and c_j is the number of occurrence of system call s_j in the input sequence Z_i.

Thus, the original trace is converted to a bag of system calls, and the ordering information of adjacent system calls in the input sequence is lost and only the frequency of each system call in the bag is preserved. Intrusion in this feature representation is defined according to frequency count of system calls.

One of the main issues in this paper is whether the bag of system calls representation, which is already popular in text classification and categorization, can effectively represent intrusion. We will show experimental results over University of New Mexico (UNM) and Massachusetts Institute of Technology Lincoln Lab (MIT LL) data in later sections for this issue. It will be shown that frequency information is effective enough to discriminate between normal sequences and abnormal sequences. As an example for this, figure 1 shows a histogram of the average frequency of selected system calls in normal sequences and abnormal sequences in UNM denial of service (DoS) trace data set (also known as stide data set). We found a similar phenomenon for other data sets including MIT LL data sets.

III. Data Sets

For experiments, we choose publicly available system call sequences from UNM and MIT LL data.

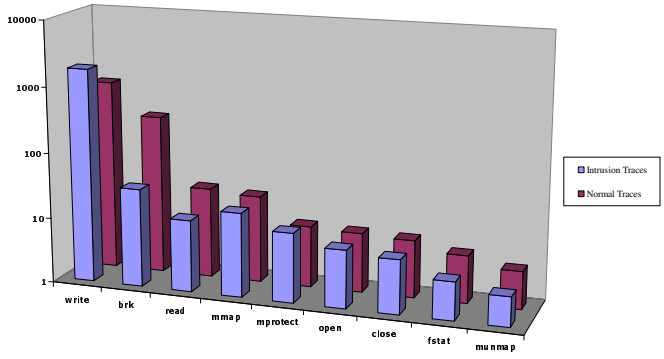


Fig. 1. Average frequency of selected system calls in normal traces and intrusion traces in UNM denial of service trace data set

A. UNM System System Call Sequences

The University of New Mexico (UNM) provides a number of system call data sets. Each data set corresponds to a specific attack or exploit. The data sets we tested are "live lpr", "live lpr MIT", "synthetic sendmail", "synthetic sendmail CERT", and "denial of service"(DoS).

In UNM system call traces, each trace is an output of one program. Sometimes, one trace has multiple processes. In such cases, we have made one sequence per process in the original trace. Thus, multiple sequences of system calls are made from one trace if the input trace has multiple processes in it. However, most traces have only one process and usually one sequence is created for each trace. Table I shows the number of original traces and the number of sequences for each data set.

There are three different mapping files in UNM call traces. One is Sun (synthetic sendmail, synthetic sendmail CERT, synthetic lpr, live lpr and live lpr.MIT) , another is Linux (live named, login, ps, inet and DoS), and the third is new Linux (synthetic ftp and xlock). There are old and new Sun mapping files but only one system call is added to the new mapping file so both can be easily converted. The Sun mapping file has a few duplicate system calls (e.g. 'fstat', 'stat', etc.), but we changed them so that each system call is unique.

B. MIT Lincoln Lab System Call Sequences

We used data sets provided by the MIT Lincoln Lab [20]. The fourth week (starting at 6/22/98) training data set of year 1998 is used for the experiments in this paper. This training data is comprised of a detailed set of data files representing the state of a particular system over eight-hour daytime periods over the course of the week beginning on 6/22/98. Of interest to this paper is the omnibus data file containing all system calls made during the collection period and the network traffic analysis file (distilled from raw network data) that identifies inbound network connection attempts.

We explain the issues with cross-indexing the data files. MIT Lincoln Labs datasets include omnibus files containing all system call traces. For each omnibus file, there is with a separate, network traffic analysis data file that indicates inbound network connections to the system. Attack attempts are logged with the network data, so labeling of the training data requires cross-indexing this file with the system call trace file. The system call trace file identifies the source of each call using the process ID. Therefore, cross-indexing requires tracking the argument to the 'exec' system call identifying the binary to be executed. Additionally, the timestamps from the network traffic analyzer do not exactly correspond to the execution timestamps from the operating system kernel. A tolerance of one second was arbitrarily chosen and seems to permit the matching of a large majority of connection attempts with their corresponding server processes run on the target system.

All processes detected that do not correspond to some network connection attempt identified in the trace are removed from consideration (since they cannot be classified), as are all calls attributed to a process ID for which an 'exec' system call is not found. The resulting data are available at `http://www.cs.iastate.edu/~dkkang/IDS_Bag/`.

IV. Experiments and Results

We use different approaches for three different types of intrusion detection experiments. The approaches will be explained at each respective section.

The data sets we have tested are "live lpr", "live lpr MIT", "synthetic sendmail", "synthetic sendmail CERT", and "denial of service attack" of UNM, and the fourth week training data set of year 1998 in MIT LL.

For the evaluation of classifiers generated in the experiment, 10-fold cross validation is used, so no training information is reused in the test stage. In 'x'-fold cross-validation, the data set is divided into x subsets of approximately equal size. One of the subsets is picked for testing and the rest subsets are used for training. In other words, a classifier is generated from 'x-1' subsets and the classifier is tested over the rest subset. This routine is applied for each of x different subsets, and then accuracy, detection rate and false positive rate are averaged respectively over each of x different subsets tested. This is to ensure that no information used for classifier generation is reused as test data.

Accuracy, detection rate, and false positive rate are defined as follows:

$$\text{accuracy} = \frac{\#\text{ of true positives} + \#\text{ of true negatives}}{\#\text{ of input sequences}}$$

$$\text{detection rate} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false negatives}}$$

$$\text{false positive rate} = \frac{\#\text{ of false positives}}{\#\text{ of true positives} + \#\text{ of false positives}}$$

TABLE I
THE NUMBER OF ORIGINAL TRACES AND GENERATED SEQUENCES IN UNM DATA SETS

Program	# of original traces	# of sequences
live lpr (normal)	1232	1232
live lpr (exploit)	1001	1001
live lpr MIT (normal)	2704	2704
live lpr MIT (exploit)	1001	1001
synthetic sendmail (normal)	7	346
synthetic sendmail (exploit)	10	25
synthetic sendmail CERT (normal)	2	294
synthetic sendmail CERT (exploit)	6	34
denial of service (normal)	13726	13726
denial of service (exploit)	1	105

A. Experimental Results on Misuse Detection

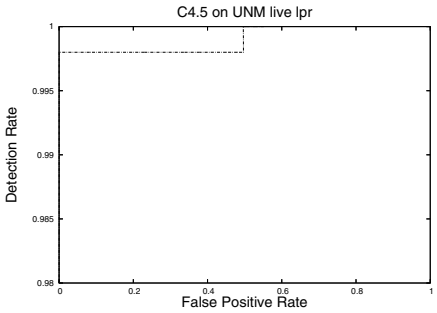
For misuse detection, we use several machine learning techniques. We tested Naive Bayes Learner of Multinomial Event Model [11], C4.5 Decision Tree Learner [10], RIPPER rule learner [12], SVM [13], and Logistic Regression Model [15]. For SVM, Sequential Minimal Optimization (SMO) [14] with a linear kernel was used for training, and for logistic regression, a multinomial logistic regression model with a ridge estimator was used. Table II shows the accuracy, detection rate, and false positive rate of the data sets we tested. The detection rate is a fraction of the intrusions identified and the false positive rate is a fraction of normal data mis-identified as intrusion.

The results in table II show that standard machine learning techniques are effective in misuse detection with simple bag of system calls representation. For example, with SMO using a linear kernel, an SVM can perfectly detect both normal and intrusion sequences in the "UNM live lpr" data set.

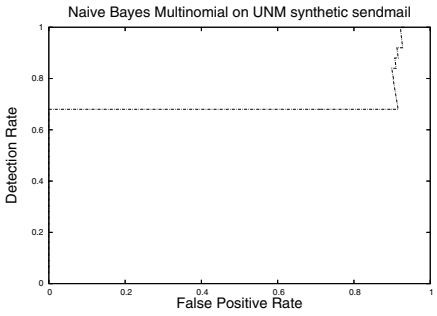
In the MIT LL results in table II, it is interesting that all machine learning algorithms got the same results on each day tested. We did not try to detect the type of intrusion and assign a corresponding score for the intrusion as was intended in the original evaluation in 1998. Instead we just tried to detect intrusion. Perhaps, the reason that all algorithms have the same results for each day is that the normal sequences and intrusion sequences in the data set are already highly different. Wednesday data set was not tested because no intrusions were in the network traffic analysis file.

One problem which is common in intrusion detection practice is that data is not quite balanced. For example, "UNM synthetic sendmail" and "UNM synthetic sendmail CERT" data sets in the table II are such data sets, and that's why their detection rate or false positive rate is not quite optimal.

Figure 2 shows the Receiver Operating Characteristic (ROC) Curve of "UNM live lpr" and "UNM synthetic sendmail" data sets using C4.5 and Naive Bayes Multinomial algorithms respectively.



(a)C4.5 decision tree induction on UNM live lpr



(b)Naive Bayes Multinomial on UNM synthetic sendmail

Fig. 2. ROC Curve of "UNM live lpr" and "UNM synthetic sendmail" data sets in misuse detection

From figure 2(a), we can see that the classifier generated from "UNM live lpr" data set is very effective because it has sufficient number of intrusion data for training. The "UNM live lpr" data set has 183 attributes, 1232 normal sequences, and 1001 intrusion sequences. From figure 2(b), standard machine learning techniques have limitations in this case, because the data sets themselves are small. The "UNM synthetic sendmail" data set has 182 attributes, 346 normal sequences, and only 25 intrusion sequences, and we

TABLE II
PERCENTAGE OF MISUSE DETECTION BASED ON 10 FOLD CROSS-VALIDATION

Program	Naive Bayes Multinomial	C4.5	RIPPER	SVM	Logistic Regression
UNM live lpr					
accuracy	83.43	99.91	99.91	100.00	99.91
detection rate	100.00	99.80	99.80	100.00	100.00
false positive rate	30.03	0.00	0.00	0.00	0.16
UNM live lpr MIT					
accuracy	54.52	99.89	99.86	99.83	99.97
detection rate	100.00	99.90	99.80	99.80	99.90
false positive rate	62.31	0.11	0.11	0.14	0.00
UNM synthetic sendmail					
accuracy	20.21	94.87	94.33	95.68	95.41
detection rate	92.00	40.00	48.00	40.00	64.00
false positive rate	84.97	1.15	2.31	0.28	2.31
UNM synthetic sendmail CERT					
accuracy	24.39	96.64	95.42	96.03	96.03
detection rate	100.00	85.29	82.35	64.70	82.35
false positive rate	84.35	2.04	3.06	0.34	2.38
UNM denial of service					
accuracy	98.70	99.97	99.96	99.98	99.97
detection rate	44.76	99.04	98.09	100.00	99.04
false positive rate	0.88	0.02	0.02	0.01	0.01
MIT LL 1998 4^{th} Week					
Monday					
accuracy	100.00	100.00	100.00	100.00	100.00
detection rate	100.00	100.00	100.00	100.00	100.00
false positive rate	0.00	0.00	0.00	0.00	0.00
Tuesday					
accuracy	99.55	99.55	99.55	99.55	99.55
detection rate	98.60	98.60	98.60	98.60	98.60
false positive rate	0.00	0.00	0.00	0.00	0.00
Thursday					
accuracy	99.73	99.73	99.73	99.73	99.73
detection rate	100.00	100.00	100.00	100.00	100.00
false positive rate	0.04	0.04	0.04	0.04	0.04
Friday					
accuracy	98.80	98.80	98.80	98.80	98.80
detection rate	89.28	89.28	89.28	89.28	89.28
false positive rate	0.00	0.00	0.00	0.00	0.00

tested the generated classifier with 10 fold cross-validation. Hence, in some folds, the machine learning algorithm did not have enough intrusion samples. Machine learning algorithm needs statistically sufficient data for each class to generate a useful classifier. But when the data is imbalanced, it cannot obtain sufficient data to generate a classifier to discriminate the classes effectively.

B. Detecting intrusion from the generated rules

One of the problems in our bag of system calls representation is that, with some machine learning algorithms, the classification cannot be done until the end of the process [18]. However, with the machine learning algorithms that generate comprehensive hypotheses, we can use very simple rules to detect a process that has exhibited intrusive behavior before it is terminated.

Figure 3 is the decision tree by C4.5 for the "UNM live lpr" data set.

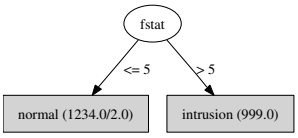


Fig. 3. C4.5 decision tree for UNM live lpr

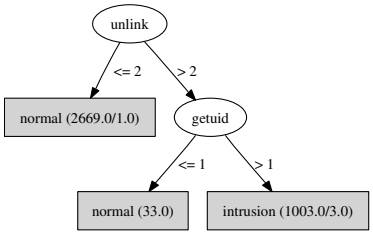


Fig. 4. C4.5 decision tree for UNM live lpr MIT

Though this simple rule does not have a perfect detection rate, it says that "we can guess the input lpr program trace is an intrusion sequence if the number of occurrences of 'fstat' system calls is more than 5". Therefore, a simple counter program that counts the number of certain system calls can detect intrusion before the process ends. However, unlike the approach that detects foreign contiguous subsequences, the counter program may not detect intrusion just after foreign subsequences are executed.

Figure 4 is the decision tree produced by C4.5 for "UNM live lpr MIT" data set. Though both the "UNM live lpr" data and the "UNM live lpr MIT" data contain intrusion snapshots by "*lprcp*" scripts, the generated rule may not always be the same because of different system environments.

The reason that this difference in frequency matters in classification is that the programs compromised by the intruder will have more codes (intrusion codes) which will be executed during the routine execution of the programs, causing a change in the distribution of system calls. In the decision trees of figure 3 and 4, it can be seen that intrusion lies under 'greater than (>)' arc. It is because adding intrusion codes in the original program increases the counts of those system calls ('fstat', 'unlink', and 'getuid') in the decision trees.

C. Experimental Results on Supervised Anomaly Detection

For supervised anomaly detection, we used one class Naive Bayes algorithm. In one class Naive Bayes, we calculate the probability distribution of the training data instead of the class label conditional probability distribution. For test sequences, we calculated symmetric Kullback-Liebler divergence [21], [22] between the learned distribution and the distribution of test sequence in bag of system calls representation. If the divergence is under a user-specified threshold θ, then the test sequence is considered to be similar to the learned distribution.

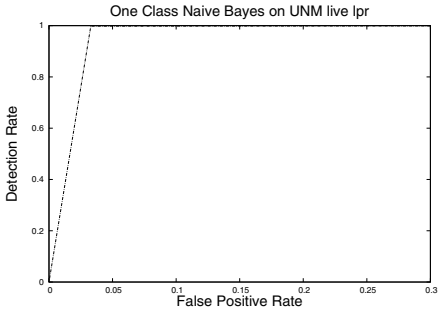


Fig. 5. ROC Curve of One class Naive Bayes on UNM live lpr ($\theta = 0.43$)

In figure 5, we show the result of the one class Naive Bayes algorithm in a bag of system calls representation on "UNM live lpr" data set.

One class Naive Bayes performs effectively on the "UNM live lpr" data set, but does not perform effectively on some of other data sets, especially when the data set is imbalanced.

D. Experimental Results on Unsupervised Anomaly Detection

In unsupervised anomaly detection, the learning algorithm assumes that the input data set is composed of normal sequences and intrusion sequences, but the sequences are not explicitly labeled. Therefore, it assumes the data distribution is a mixture of the distribution of normal seqeunces and the intrusion seqeunces. We use k-Means clustering with k set to 2 for clustering normal and intrusion distributions. We evaluated the clustering based approach on "UNM live lpr" and "UNM synthetic sendmail" data. The results are shown in table III.

From the results in the table, k-means clustering is effective in unsupervised anomaly detection on 'UNM live lpr' data but not on 'UNM synthetic sendmail' data.

V. Summary and Discussion

In this paper, we have explored the use of a simple *bag of system calls* representation of system call sequences for intrusion detection. We constructed decision tree, Naive Bayes, decision list, and SVM and Logistic Regression classifiers for misuse detection. We constructed one class Naive Bayes algorithm and K-Means clustering for anomaly detection. In addition to the fact that we can use those standard machine learning methods, the proposed 'bag of system calls' representation has significant computational advantages over other approaches that have been reported in the literature.

Results of our experiments using widely used benchmark data sets - the University of New Mexico (UNM) and MIT Lincoln Lab (MIT LL) system call sequences show that the

TABLE III
K-MEANS CLUSTERING RESULTS OF UNSUPERVISED ANOMALY DETECTION EXPERIMENTS IN PERCENTAGE.

Program	Accuracy	Detection Rate	False Positive
UNM live lpr	99.28	100.00	1.29
UNM synthetic sendmail	80.3235	40.00	16.76

performance of the proposed approach in terms of detection rate and false positive rate is comparable or superior to that of previously reported data mining approaches to misuse detection. In particular, as shown in table II, the proposed methods achieve nearly 100% detection rate with almost 0% false positive rate on all the data sets studied with the exception of two synthetic data sets ('UNM synthetic sendmail' and 'UNM synthetic sendmail CERT'). It is important to note that the reported performance measures were estimated using 10 fold cross-validation which ensures no overlap between training data and test data.

A. Discussion

When compared with the widely used fixed-length contiguous subsequence models, the *bag of system calls* representation explored in this paper may seem somewhat simple. It may be argued that much more sophisticated models that take into account the identity of the user or perhaps the order in which the calls were made. But our experiments show that a much simpler approach may be adequate in many scenarios. The results of experiments described in this paper show that it is possible to achieve nearly perfect detection rates and false positive rates using a data representation that discards the relationship between system call and originating process as well as the sequence structure of the calls within the traces.

Forrest et. al. [5], [18] showed that it is possible to achieve accurate anomaly detection using fixed-length contiguous subsequence representation of input data. In their approach, the detector will find anomalous subsequences right after they are executed depending on user-specified thresholds. The proposed '*bag of system calls* representation has advantages that learning is faster, memory requirements are significantly lower, and simple counter program can discriminate normal sequences and abnormal sequences very quickly, before the process is terminated.

In these respects, a bag of system call representation is very suitable for protecting well known attacks and trivially modified attacks for IDS under time and space constraints. If the IDS needs to be built in real-time and the built system must be as light as possible to be able to work over limited resources, our approach will be a perfect fit because the generated IDS is simple and powerful to detect well known attacks. However, if the attacker knows the intrusion detection mechanism, our approach can be deceived by mimicry attacks [23]. Our future work will be focused on addressing this problem.

B. Related Work

Warrender, Forrest, and Pearlmutter [18] have presented several intrusion detection methods based upon system call trace data. They tested a method that utilizes sliding windows to determine a database of *normal* sequences to form a database for testing against test instances. They then used a similar method to compare windows in the test instances against the database and classify instances according to a function of the similarity of these sequences to those in the *normal* sequence database. The function requires sequential analysis of a window of system calls for each call made by a process. This requires the maintenance of a large database of *normal* system call trace sequences.

The same authors have described a rule-based classification method that requires alterations to the training data to learn. This model involves prediction of the next system call to be made by a process given some number of calls made immediately before. This method requires enumeration of all unique system call traces within a given program. This is quite demanding on a learner, especially in a situation where the datasets are quite large indeed. Even the space requirements are quite large relative to the input dataset. Finally, classification time is high for such methods because (in the worst case) each rule needs to be checked for each input instance.

Warrender et al. have presented Hidden Markov Model (HMM) methods for intrusion detection. Although this method does not require modification of the input dataset, it does require individual examination of each dataset to determine the optimal HMM to attempt to learn in each case. While this requirement does not seem overly demanding, we would prefer a method which allows classification of multiple input datasets in the same format if possible. Additionally construction of accurate HMM models can be quite demanding in terms the amount of training data as well as computational effort. Warrender, et al. observe that, for a process that makes S system calls, S states (and thus $2S^2$ values) must be computed. Datasets of interest in practice contain large amounts of processes (eight hours per day worth in the case of the MIT Lincoln Labs datasets), and each process makes a large number of system calls throughout its lifetime. Computing even polynomially many values for each instance becomes a problem at this scale.

Normalized frequency of audit data was used in SRI NIDES [24]. In NIDES, probability distribution of long

term behavior of a program is generated and maintained as its profile. For detecting the anomalous behavior of the program, the profile is compared with short term behavior of the program, which is also maintained as probability distribution, using a statistical test similar to χ^2 test. The behavior of a program is characterized by its audit data such as file access, CPU usage, etc. We maintain the raw count of system calls that are sequentially observed from the program as its profile, but this approach can be applied to other types of audit data. In some machine learning algorithms, raw counts are normalized and statistically compared with new behavior of the program. The Naive Bayes learning algorithm, which is one of the learning algorithms reported in this study, generates class-conditional probability distributions and prior distributions of the raw counts and statistically compares them with new distribution from the new behavior of the program. Moreover, as we showed, our profile representation can be used effectively with various machine learning algorithms.

One of the most popular rule induction techniques used in IDS is Repeated Incremental Pruning to Produce Error Reduction (RIPPER) rule learning algorithm [12]. Lee et al. [25] used RIPPER on a set of substrings of length 7 generated by the sliding window from *sendmail* system call traces. The generated rules are based on the insight that intrusion can be captured from the fixed-length substrings. For example, the rule *'normal:* $p_2 = 104, p_7 = 112$*'* means *'if* p_2 *is 104 and* p_7 *is 112 then the substring is normal'*. This approach, as in the case of *STIDE*, employs a user-supplied threshold to determine if the input trace is normal or intrusive. We applied RIPPER on a bag of system calls representation, and we obtained rules based on counts such as *'(count(fcntl)* $\geq$ *1) and (count(rename)* $\leq$ *0) and (count(read)* $\geq$ *5)* $\rightarrow$ *class=intrusion'* where *count(X)* returns the number of occurrence of system call X in the input trace. The rules generated by our method apply to the entire system call trace (as opposed to fixed length substring of traces). In our case, the relevant thresholds are learned directly from the training data, thereby avoiding the necessity of user-supplied thresholds.

References

[1] D. E. Denning, "An intrusion-detection model," *IEEE Trans. Softw. Eng.*, vol. 13, no. 2, pp. 222–232, 1987.

[2] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Tech. Rep. 99-15, Chalmers Univ., Mar. 2000.

[3] A. Ghosh and A. Schwartzbard, "A study in using neural networks for anomaly and misuse detection," in *8th USENIX Security Symposium*, (Washington, D.C.), pp. 141–151, 1999.

[4] B. Mukherjee, H. Heberlein, and K. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994.

[5] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, p. 120, IEEE Computer Society, 1996.

[6] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *IEEE Symposium on Security and Privacy*, pp. 120–132, 1999.

[7] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and R. Lutz, "A software fault tree approach to requirement analysis of an intrusion detection system," in *Symposium on Requirements Engineering for Information Security*, 2001.

[8] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data," *Data Mining for Security Applications*, 2002.

[9] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[10] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[11] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[12] W. W. Cohen, "Fast effective rule induction," in *Proc. of the 12th International Conference on Machine Learning* (A. Prieditis and S. Russell, eds.), (Tahoe City, CA), pp. 115–123, Morgan Kaufmann, July 9–12, 1995.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[14] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods: support vector learning*, pp. 185–208, 1999.

[15] S. L. Cessie and J. V. Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.

[16] C. M. Bishop, *Neural networks for pattern recognition*. Oxford University Press, 1996.

[17] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[18] C. Warrender, S. Forrest, and B. A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *IEEE Symposium on Security and Privacy*, pp. 133–145, 1999.

[19] K. M. C. Tan and R. A. Maxion, ""Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, p. 188, IEEE Computer Society, 2002.

[20] R. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman, "Results of the darpa 1998 offline intrusion detection evaluation.," in *Recent Advances in Intrusion Detection*, 1999.

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, pp. 79–86, 1951.

[22] M. Li and P. M. B. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*. Berlin: Springer-Verlag, 1993.

[23] D. Wagner and P. Soto, "Mimicry attacks on host based intrusion detection systems," in *Proc. Ninth ACM Conference on Computer and Communications Security*, 2002.

[24] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and A. Valdes, "Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES)," Tech. Rep. SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA, May 1995.

[25] W. Lee and S. Stolfo, "Data mining approaches for intrusion detection," in *Proceedings of the 7th USENIX Security Symposium*, (San Antonio, TX), 1998.