Learning Patterns from Unix Process Execution Traces for Intrusion Detection[1]

Wenke Lee and Salvatore J. Stolfo

Computer Science Department
Columbia University
500 West 120th Street
New York, NY 10027
{wenke,sal}@cs.columbia.edu

Philip K. Chan

Computer Science
Florida Institute of Technology
Melbourne, FL 32901
Pkc@cs.fit.edu

Abstract

In this paper we describe our preliminary experiments to extend the work pioneered by Forrest (see Forrest et al. 1996) on learning the (normal and abnormal) patterns of Unix processes. These patterns can be used to identify misuses of and intrusions in Unix systems. We formulated machine learning tasks on operating system call sequences of normal and abnormal (intrusion) executions of the Unix *sendmail* process. We show that our method can accurately distinguish all abnormal executions of *sendmail* from the normal ones provided in a set of test traces. These preliminary results indicate that machine learning can play an important role by generalizing stored sequence information to perhaps provide broader intrusion detection services. The experiments also reveal some interesting and challenging problems for future research.

Introduction

Misuse and intrusion of computer systems has been a pervasive problem ever since computers were first invented. With the rapid deployment of network systems, intrusions have become more common, their patterns more diverse, and their damages more severe. As a result, much effort has been devoted to the problem of detecting intrusions as quickly as possible.

There are two basic approaches to intrusion detection (see (Forrest et al. 1996)):

- Misuse Intrusion Detection: known patterns of (past) intrusions are used to identify intrusions as they happen, as in COAST (Kumar and Spafford 1995) and STAT (Ilgun et al. 1995).
- Anomaly Intrusion Detection: recognizes behaviors that deviate from (recorded) normal behavior (as in (Forrest et al. 1996)).

Since intruders are constantly inventing new (hence a priori unknown) attacks, using only the misuse intrusion detection method will likely be inadequate in fully protecting any computer system. Some

[1] This research is supported in part by grants from DARPA (F30602-96-1-0311), NSF (IRI-96-32225 and CDA-96-25374), and NYSSTF (423115-445).

systems, for example the IDES system (Lunt et al. 1992), use both approaches.

Anomaly intrusion detection usually involves the use of profiles for user behavior or privileged processes. When a user profile is used, intrusion is detected when a user behaves out of his/her (normal) character according to statistical profiles (Lunt et al. 1992) or induced patterns of behavior (Teng et al. 1990). The main difficulty with this approach is that generation of user profiles requires potentially large amount of audit trail data about actions, which can change dynamically over time, of each user or population of users. More recently, researchers have tried instead to determine the normal behavior for privileged processes (those that run as root). Ko, Fink and Levitt (Ko et al. 1994) used a program specification language to formally specify the normal behavior of a program as the collection of allowed operations (the system calls and their parameters). Forrest et al. (Forrest et al. 1996) introduced a novel and simpler method. They gathered the traces of normal runs of a program and analyzed the "local (short) range ordering of system calls". They discovered that these local orderings "appears to be remarkably consistent, and this suggests a simple definition of normal behavior". The key idea here is to build a "normal" database that contains all possible short sequences (e.g., of length 11) of system calls for each program (*sendmail*, *lpr*, etc.) that needs to be guarded. The normal database is then used to examine the behavior of a running program (e.g., an instance of *sendmail*). If the total number (or percentage) of abnormal sequences, which are those that can not be found in the normal database, is above an empirically established threshold value, then the current run is flagged as abnormal, i.e., a misuse or intrusion is detected.

We have been studying the application of machine learning (and meta-learning (Chan & Stolfo 1993)) to fraud and intrusion detection in financial information systems (Stolfo et al.1997). Here we consider the means of applying our technologies to explore other closely related tasks. Stephanie Forrest has provided us a set of traces of the *sendmail* program to experiment with. These traces were used in the experiments reported in (Forrest et al. 1996). Our goal here is to investigate whether a machine learning approach can be used to learn the normal and/or abnormal patterns from the data, thus generalizing the "rote" learning of static "normal only" sequence information. More importantly, we want to study whether our approach can produce more accurate and/or more efficient intrusion detection capabilities to perhaps improve upon what Forrest et al. have reported.

Experiments on the *sendmail* System Call Data

System Call Data

We have obtained two sets of *sendmail* system call data. The procedures of generating these traces are described in (Forrest et al. 1996). Each file of the trace data has two columns of integers, the first is the process ids and the second is the system call "numbers" (see Table 1). These numbers are indices into a lookup table of system call names. For example, the number "5" represents system call "*open*". Since *sendmail* can *fork*, its child processes are traced separately, but their traces are all included in the trace of the current run of *sendmail*. The set of traces include:

- Normal traces: a trace of the *sendmail* daemon and a concatenation of several invocations of *sendmail*.
- Abnormal traces: 3 traces of the *sscp* intrusion, 2 traces of the *syslog-remote* intrusion, 2 traces of the *syslog-local* intrusion, 2 traces of the *decode* intrusion, 1 trace of the *sm5x* intrusion and 1 trace of the *sm565a* intrusion. These are the traces of (various kinds of) "abnormal" runs of the *sendmail* program.

The *sendmail* daemon deals with incoming mail and all other processes deal with outgoing mail.

pids	282 282 …	291 291…
system calls	4 2 66 66 4 138 66 5 23 45 4 27 …	155 104 106 105 104 104 106 56 19 155 83 155 …

Table 1. System Call Data. Each file has two columns, the pids and the system call numbers.

Pre-processing to Create Training Data

Intuitively, the temporal ordering of system calls are important characteristics of a program's normal behavior. The simplest way of representing the (short distance) temporal information is to use a sliding window to create sequences of consecutive system calls so that system calls that are close to each other (in time steps) are in a single unit. Following (Forrest et al 1996), a sliding window of length 11 seemed to give the best predictive performance. Therefore, we also use a sliding window of length 11 with sliding (shift) step of 1 to create sequences of system calls from the traces.

We first use a sliding window to scan the normal traces (of the *sendmail* daemon and *sendmail*) and create a list of unique sequences of system calls, 1,082 in total. We call this list the "normal" list. Next, we scan each of the intrusion traces. For each sequence of 11 adjacent system calls, we first look it up in the "normal" list. If a match can be found then the sequence is labeled as "normal". Otherwise it is labeled as "abnormal". See Table 2 for an example of the labeled sequences. Needless to say all sequences in the normal traces are labeled as "normal". It should be noted that for an intrusion trace not all the sequences are "abnormal" since the illegal activities only occur in several places within a trace.

System Call Sequences (length 11)	Class Labels
4 2 66 66 4 138 66 5 23 45 4	"normal"
…	…
104 106 105 104 104 106 56 19 155 83 155	"abnormal"
…	…

Table 2. Pre-processed System Call Data. System call sequences of length 11 are labeled as "normal" or "abnormal".

Experimental Setup

We applied RIPPER (Cohen 1995), a rule learning program, to our training data. RIPPER is fast and generates concise rule sets. It is very stable and has shown to be consistently one of the best algorithms in our past experiments (see Stolfo 1997).

We formulate our learning task as followings:

- Each record has 11 (positional) attributes, p_5, p_4, …, p0, p1, … p5, one for each of the positions in a system call sequence of length 11; plus a class label, "normal" or "abnormal".
- The values of each attribute are specified as symbolic rather than numerical.
- The training data is composed of all normal sequences, plus the abnormal sequences from 2 traces of the *sscp* intrusion, 1 trace of the *syslog-local* intrusion, and 1 trace of the *syslog-remote* trace.
- The testing data is all the sequences (normal and abnormal) in the intrusion traces not included in the training data.

RIPPER outputs a set of if-then rules for the "minority" classes, and a default "true" rule for the

remaining class. The following exemplar RIPPER rules were generated from the system call data:

[covers 84 positive and 0 negative examples; here positive=”abnormal” and negative=”normal”]
abnormal:- p_5=’112’, p1=’112’, p3=’128’.
[meaning: if p_5 and p1 are 112 (*vtrace*) and p3 is 128 (*flock*) then the sequence is “abnormal”]

[covers 75 positive and 0 negative examples]
abnormal:- p0=’128’, p2=’112’.
[meaning: if p0 is 128 and p2 is 112 then the sequence is “abnormal”]
…
[covers 4188 positive and 0 negative examples; here positive=”normal” and negative=”abnormal”]
normal:- true.
[meaning: if none of the above, the sequence is “normal”]

The RIPPER rules can be used to predict whether a sequence is “abnormal” or “normal”. But what the intrusion detection system needs to know is whether the trace being tested is an intrusion or not. Can we say that whenever there is an “abnormal” sequence in the trace, it is an intrusion? It depends on the accuracy of the rules when classifying a sequence as “abnormal”. Unless it is close to 100%, it is unlikely that a predicted “abnormal” sequence is always part of an intrusion rather than just an error. Unlike the cases of fraud detection in mobile phone or credit card transactions, where false positives (false alarms) can be resolved by human intervention, false alarms of a trace here may result in a program being terminated. This is a highly undesirable outcome since some data of the program can be lost permanently.

Post-processing for Intrusion Detection

We use the following post-processing scheme to detect whether the trace is an intrusion based on the RIPPER predictions of its constituent sequences:

1. Use a sliding window of length $2n+1$, e.g., 7, 9, 11, 13, etc., and a sliding (shift) step of n, to scan the predictions made by RIPPER.
2. For each of the regions (of RIPPER predictions) generated in Step1, if more than n predictions are “abnormal” then the current region of predictions is an “abnormal” region. (Note that n is an input parameter)
3. If the percentage of “abnormal” regions (of RIPPER predictions) is above a threshold value, say 5%, then the trace is an intrusion.

This scheme is an attempt to filter out the spurious mistakes (wrongly classified “abnormal” sequences). The intuition behind this scheme is that when an intrusion actually occurs, it generates a number of abnormal system calls, and as a result, the neighboring sequences of system calls will not match the normal sequences. Therefore the RIPPER rules would predict that the majority of the adjacent sequences are “abnormal”. However, the prediction errors tend to be isolated, i.e., the (false) abnormal predictions are sparse. Note that we could have used a sliding step of 1, but obviously, a step greater than 1 is much more efficient. In fact, a sliding step of n ensures that a “majority” group (greater than n occurrence) of “abnormal” predictions of any region of $2n+1$ consecutive predictions will not be missed as the window slides.

In (Forrest et al. 1996), the percentage of the mismatched sequences (out of the total number of matches (lookups) performed for the trace) is used to distinguish normal from abnormal. The “mismatched”

sequences are the "abnormal" sequences in our context. Our scheme is different in that we look for "abnormal regions" that contains more "abnormal" sequences than the "normal" ones, and calculate the percentage of abnormal regions (out of the total number of regions). Our scheme is more sensitive to the temporal information, and is less sensitive to noise (mistakes).

Results

We now analyze the results of our experiments. We only report here the experiments that show results that improve upon other methods. We demonstrate that our machine learning approach is indeed viable in detecting misuse and intrusions. We also highlight the limitations that we hope to overcome in our future research.

Recall that RIPPER only outputs rules for the "minority" class. For example, in our experiments, if the training data has fewer "abnormal" sequences than the "normal" ones, the output RIPPER rules can be used to identify "abnormal" sequences, and the default (everything else) prediction is "normal". We want to compare the results of using different distributions of "abnormal" versus "normal" in the training data. We conjecture that a set of specific rules for "normal" sequences can be used as the "identity" of a program, and thus can be used to detect any known and unknown intrusions (anomaly intrusion detection). Whereas having only the rules for "abnormal" sequences only gives us the opportunity to identify known intrusions (misuse intrusion detection).

Thus the training data needs to have both "normal" and "abnormal" sequences. Obviously we need to include all the unique normal sequences (total 1,082). The abnormal sequences are taken from *sscp-1*, *sscp-2*, *syslog-local-1*, and *syslog-remote-1*. We compare the results of the following experiments that have different distributions of "abnormal" versus "normal" sequences in the training data:

1. Experiment A: 1 copy of all unique normal sequences and 1 copy of the abnormal sequences.
2. Experiment B: 5 copies of all unique normal sequences and 1 copy of the abnormal sequences. (Note that the frequency distribution is artificially biased here as well as in the next two experiments)
3. Experiment C: 4 copies of all unique normal sequences and 3 copies of the abnormal sequences.
4. Experiment D: 3 copies of all unique normal sequences and 3 copies of the abnormal sequences.

Each copy of the abnormal sequences has 1,315 sequences. Therefore, Experiment A and D output classifiers with specific rules for the "normal" sequences, whereas Experiment B and C generate classifiers with specific rules for the "abnormal" sequences. Also note that Experiment B has a very skewed distribution (15% "abnormal").

We test the performance of the RIPPER generated classifiers on every intrusion trace by supplying all the sequences (abnormal and normal) of the trace to the classifiers. The post-processing scheme, with a sliding window of length 9, is applied to the predictions of the classifiers. Note that the window length here specifies the size of the regions of predictions, which can be different from the length of the sequences of system calls. We also test the classifiers on the list of 1,082 unique normal sequences. Table 3 shows the anomaly detection results of these experiments alongside the results from Forrest et al. (1996).

From Table 3, we can see that the classifier from Experiment A, which has rules for "normal" sequences, is not acceptable because it classifies every trace (including *sendmail*) as an intrusion. This is due to too few examples of "normal" and "abnormal" in the training data (each unique "normal" sequence only appears once). The classifier from Experiment B, which has rules for "abnormal"

sequences, is an improvement. It predicts correctly on *sscp-3*, *syslog-local-2*, and *syslog-remote-2*, but misses *decode-1*, *decode-2*, *sm565a*, *sm5x*, because the percentages are below the threshold value of 5%. The classifier from Experiment C, which also has rules for "abnormal" sequences, does a little bit better than the classifier from Experiment B. Therefore a more balanced class distribution does help improve the performance (confirming our results on other fraud data (Stolfo et al. 1997)). It is important to note that the classifiers from Experiment B and C perform quite well on known intrusions, i.e., *sscp-3*, *syslog-local-2*, and *syslog-remote-2*, because the training data includes the abnormal sequences from the traces of the same types of intrusions. But they perform relatively poorly on unknown intrusions (the abnormal sequences of the traces of these types of intrusions are not in the training data), i.e., *decode-1&2*, *sm565a*, and *sm5x*. This confirms our conjecture that classifiers with rules for (known) "abnormal" sequences are only good for detecting known intrusions and hence don't generalize to other "unseen" intrusions. The classifier from Experiment D, which has rules for "normal", has the best performance. It correctly classifies every trace of known and unknown intrusions. This also confirms our conjecture that classifiers with rules for the "normal" sequences can be used for anomaly intrusion detection, thus generalizing the notion of normalcy. The results from Forrest et al. (1996) showed that their methods require a very low threshold in order to correctly detect the *decode* and *sm565a* intrusions. The results from our experiments showed that our approach generates much "stronger signals" of anomalies from the intrusion traces. It should also be noted that Forrest et al. (1996) did not use the sequences from the intrusion traces except for testing, whereas the training data in our experiments contains (abnormal) sequences from several intrusion traces.

Traces	Forrest et al. (percentage of abnormal sequences)	Experiment D (percentage of abnormal regions)	Experiment C (percentage of abnormal regions)	Experiment B (percentage of abnormal regions)	Experiment A (percentage of abnormal regions)
*sscp-1	5.2	47.8	44.5	41.2	46.7
*sscp-2	5.2	48.3	43.8	41.6	47.2
*sscp-3	5.2	48.3	43.8	41.6	47.2
syslog-remote-1	5.1	47.1	32.1	29.1	60.8
syslog-remote-2	1.7	42.4	28.6	25.4	56.2
syslog-local-1	4.0	27.9	19.3	15.5	52.8
syslog-local-2	5.3	38.5	22.3	18.0	52.0
*decode-1	0.3	7.6	4.6	3.5	40.9
*decode-2	0.3	7.6	4.9	3.0	41.1
sm565a	0.6	15	5	0	33.3
sm5x	2.7	22.8	6.6	3.6	43.3
*sendmail	0	0	0	0	50.9

Table 3. Comparing Detection of Anomalies. Forrest et al. (1996) reported *sscp* and *decode* each as a single trace, whereas we report here each available trace of these intrusions. *sendmail* is the list of all unique normal sequences of system calls.

We also vary the sliding window length (the length of the regions) in our post-processing scheme. We have used lengths of 7, 11, and 13, and see very little change (less than half a percentage point) in the percentage of "abnormal" regions for each trace. This verifies our intuition of the post-processing scheme: when an intrusion occurs, there tends to be a high concentration of "abnormal" sequences.

Discussion and Future Work

The most important lesson we learned from these experiments is that in order to detect anomalous behavior (where the nature of the intrusion is unknown), it is imperative that a model of the normal behavior of the program be used. This confirms the intuition of Forrest et al. (1996) as well. For real-time detection systems to work efficiently, the models have to be simple and efficient. Forrest et al. (1996) demonstrated a novel and simple approach of using short sequences of system calls to define the normal behavior of a program. They constructed, for the *sendmail* program, a database of ~1,500 entries of legal system call sequences. The system call sequences from a run of *sendmail* can be compared with the entries of the database. The percentages of mismatches are used to classify whether the current run is an intrusion. Their experiments show that the traces from other programs (e.g., *ls*, *ping*, *pine*, etc.) have high percentages of mismatches. This result supports the claim that the short sequences of system calls are indeed program-specific. The weakness of their model may be that the recorded (rote learned) sequence database may be too specific. Here we note that RIPPER was able to generalize this sequence information to a set of rules. We hence conjecture that these RIPPER rules may be able to cover normal sequences from other traces of the same program, i.e., *sendmail*, not yet seen. Future experiments will include unseen normal sequences in the test data to evaluate the generality of RIPPER rules on normal behavior of the same program.

However, both approaches suffer a common drawback: neither can predict that an intrusion is about to occur or is actually unfolding. Both approaches can detect that an intrusion has occurred after analyzing the trace of program execution. Although we can imagine that, in real-time intrusion detection, if the percentage of mismatches (abnormal sequences) is high enough mid-way through the execution, we can determine that it is an intrusion. But this does not actually solve the problem. What is needed here is a model that includes the stronger temporal relations, e.g., y follows x in time steps d, of the sequences of system calls. Such a model can predict what should follow after having seen past system calls. Such models of both the normal behavior and the known intrusions can be used in real-time detection. If the temporal relations of the current trace match that of a known intrusion, then the current trace is an intrusion; otherwise if at some point the temporal relations start to deviate from the normal ones, it may also be flagged as an intrusion. In (Oates and Cohen 1996), MSDD was introduced as an algorithm that finds dependency rules for patterns of values that occur in multiple streams of categorical data. We plan to apply MSDD and other related algorithms to the system call data.

The success of these approaches depends upon the fact that the normal sequences are nearly exhaustive in characterizing all normal behaviors for a program. This points to the difficulties of our tasks: we need to gather enough trace data to provide the full coverage of normalcy. We need to have trace data for other programs (e.g., *ftp*) to verify our approach as well. Another related issue is that it may be favorable to also have information on the resource access patterns of a program, so that the cost of damage by abnormal calls can be estimated. With the cost information, certain traces can be detected as intrusions (and thus terminated) once a high-cost abnormal system call sequence is encountered.

Conclusions

We applied a machine learning approach to learn normal and abnormal patterns of program behavior from its execution trace to generalize upon the method introduced in (Forrest et al. 1996). The resultant normal patterns (classifiers) are shown to be able to accurately detect anomalous intrusions. Our experiments demonstrate that machine learning can indeed play an important role in intrusion detection of computer systems. Much more research needs to be pursued in order to build a system that can much

more rapidly and correctly detect intrusions.

Acknowledgements

We are very grateful to Stephanie Forrest and Steven A. Hofmeyr, both of University of New Mexico, for providing us the system call data and explaining the details of their experiments. We also wish to thank David Wei Fan of Columbia University for his helpful inputs.

References

Philip K. Chan and Salvatore J. Stolfo (1993), Toward Parallel and Distributed Learning by Meta-Learning, in *Working Notes of AAAI Work. Knowledge Discovery in Databases* (pp. 227-240)

William W. Cohen (1995), Fast Effective Rule Induction, in *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, Morgan Kaufmann

Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff (1996), A Sense of Self for Unix Processes, in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA (pp.120-128)

Tim Oates and Paul R. Cohen (1996), Searching for Structure in Multiple Streams of Data, in *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 346-354)

C. Ko, G. Fink, and K. Levitt (1994), Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring, in *Proceedings of the 10th Annual Computer Security Applications Conference* (pp. 134-144)

Koral Ilgun, Richard A. Kemmerer and Phillip A. Porras (1995), State Transition Analysis: A Rule-Based Intrusion Detection Approach, in *IEEE Transactions on Software Engineering*, 21(3)

S. Kumar and E. H. Spafford (1995), A Software Architecture to Support Misuse Intrusion Detection, in *Proceedings of the 18th National Information Security Conference* (pp. 194-204)

T. Lunt, A. Tamaru, F, Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes and T.Garvey (1992), A Real-time Intrusion Detection Expert System (IDES) – final technical report. Computer Science Library, SRI International, Menlo Park, California

Salvatore J. Stolfo, David W. Fan, Wenke Lee, Andreas Prodromidis and Phil K. Chan (1997), Credit Card Fraud Detection Using Meta-Learning: Issues and Initial Results, Technical Report CUCS-008-97, Computer Science Department, Columbia University

H. S. Teng, K. Chen and S. C. Lu (1990), Security Audit Trail Analysis Using Inductively Generated Predictive Rules, in *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, Piscataway, New Jersey