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* program. We show that our methods 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.

Keywords

Machine learning, intrusion detection, execution trace, Unix system call

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 damage 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:

- Misuse Intrusion Detection: known patterns of (past) intrusions are used to identify intrusions as they happen, as in (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 systems, for example the IDES system (Lunt et al. 1992), use both approaches.

The goal of anomaly detection is to produce statistical measures or rules, from user audit trails, that represent the "normal" usage patterns (profiles). An audit trail that shows a large deviation (above a threshold value) from the user's recorded profile can be flagged as "abnormal". The main difficulties with this approach are that user behaviors can change dynamically and frequently in many environments, and establishing the right (deviation) threshold value requires ad hoc fine tuning. More recently, researchers have tried instead to determine the normal behavior for privileged processes (those that run as root). Forrest et al. (Forrest et al. 1996) introduced a novel and simple method. They gathered the traces of

[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).

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" (Forrest et al. 1996, page 121). 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 possibly 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 with 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. 1996) has reported.

Experiments on the *sendmail* Traces

We conducted two sets of experiments. First, sequences of n consecutive system calls were extracted from *sendmail* traces and supplied to a machine learning algorithm to learn the patterns of "normal" and "abnormal" sequences. These patterns can then be used to examine a new trace and determine whether it contains sufficient abnormal sequences to be identified as an intrusion (anomaly). In the second set of experiments, the goal of the machine learning tasks was to generate rules that predict: 1) the nth system call given the n-1 preceding system calls; 2) the middle system call in a sequence of n system calls. These rules of normal *sendmail* system calls can be used to analyze a new trace to detect violations, that is, system calls appearing "out of place". Again, a large number of violations suggest an intrusion. Since (Forrest et al. 1996) indicated that sequence length 11 gave the best performance (in terms of detecting intrusions) in their experiments, we also used sequence length 11 in our experiments in order to compare the results. We also used sequence length 7 to repeat the same experiments so that we can learn the effects of sequence length on the performance of our models.

System Call Data

We have obtained two sets of *sendmail* system call data. The procedures of generating these traces were 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 the *sendmail* program.
- 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.

process ids	282 282 ...	291 291...
system calls	4 2 66 66 ...	155 104 106 105 ...

Table 1. System Call Data: each file has two columns, the process ids and the system call numbers.

In the following, we first describe in detail the experiments on learning the patterns of system call sequences, we then discuss the experiments on learning the system call prediction rules, and compare the results of the two sets of experiments in terms of their effectiveness in intrusion detection.

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.

We first use a sliding window to scan the normal traces and create a list of unique sequences of system calls. We call this list the "normal" list. Next, we scan each of the intrusion traces. For each sequence of system calls, we first look it up in the normal list. If an exact match can be found then the sequence is labeled as "normal".

Otherwise it is labeled as "abnormal". Needless to say all sequences in the normal traces are labeled as "normal". See Table 2 for an example of the labeled sequences. It should be noted that an intrusion trace contains many normal sequences in addition to the abnormal sequences since the illegal activities only occur in some places within a trace.

System Call Sequences (length 7)	Class Labels
4 2 66 66 4 138 66	"normal"
...	...
5 5 5 4 59 105 104	"abnormal"
...	...

Table 2. Pre-processed System Call Data. System call sequences of length 7 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 formulated the learning tasks as followings:

- Each record has n positional attributes, p_1, p_2, ..., p_n, one for each of the system calls in a sequence of length n; plus a class label, "normal" or "abnormal".
- The training data is composed of normal sequences taken from 80% of the normal traces, 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* intrusion.
- The testing data includes both normal and abnormal traces not used 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:

normal:- p_2=104, p_7=112.
[meaning: if p_2 is 104 (*vtimes*) and p_7 is 112 (*vtrace*) then the sequence is "normal"]

normal:- p_2=104, p_7=104.
[meaning: if p_2 and p_7 are 104 (*vtimes*) then the sequence is "normal"]
...

abnormal:- true.
[meaning: if none of the above, the sequence is "abnormal"]

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 analyzed is an intrusion or not. Can we say that whenever there is a predicted 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 trace rather than just an error of the RIPPER rules.

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

This scheme is an attempt to filter out the spurious prediction errors. The intuition behind this scheme is that when an intrusion actually occurs, the majority of adjacent system call sequences are abnormal; whereas the prediction errors tend to be isolated and 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 l ensures that a "majority" group (more than l occurrences) of "abnormal" predictions of any region of $2l+1$ 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 (errors).

Results and Analysis

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 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).

We compare the results of the following experiments that have different distributions of abnormal versus normal sequences in the training data:

1. Experiment A: 46% normal and 54% abnormal. Sequence length is 11.
2. Experiment B: 46% normal and 54% abnormal. Sequence length is 7.
3. Experiment C: 46% abnormal and 54% normal. Sequences length is 11.
4. Experiment D: 46% abnormal and 54% normal. Sequences length is 7.

Traces	Forrest et al.	A	B	C	D
***sscp-1**	5.2	41.9	32.2	40.0	33.1
***sscp-2**	5.2	40.4	30.4	37.6	33.3
*sscp-3	5.2	40.4	30.4	37.6	33.3
syslog-remote-1	5.1	30.8	21.2	30.3	21.9
syslog-remote-2	1.7	27.1	15.6	26.8	16.5
syslog-local-1	4.0	16.7	11.1	17.0	13.0
syslog-local-2	5.3	19.9	15.9	19.8	15.9
*decode-1	0.3	4.7	2.1	3.1	2.1
*decode-2	0.3	4.4	2.0	2.5	2.2
sm565a	0.6	11.7	8.0	1.1	2.1
sm5x	2.7	17.7	6.5	5.0	3.0
*sendmail	0	1.0	0.1	0.2	0.3

Table 3. Comparing Detection of Anomalies. The column Forrest et al. is the percentage of the abnormal sequences of the traces. Columns A, B, C, and D are the percentages of abnormal regions (measured by the respective experiment) of the traces. (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 20% normal traces not used in the training data. Traces in bold were included in the training data, the other traces were in testing data only.

The normal sequences were taken from 80% of the normal *sendmail* traces. These sequences constituted about 90% of all possible unique normal sequences in the entire set of normal *sendmail* traces. The abnormal sequences were taken from intrusion traces *sscp-1*, *sscp-2*, *syslog-local-1*, and *syslog-remote-1*. We tested 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. We also tested the classifiers on the remaining 20% normal *sendmail* traces not used in the training data. The post-processing scheme, with a sliding window (region) of length 9, is applied to the predictions of the classifiers. Each experiment was repeated 10 times, each using normal sequences from a different subset (80%) of the normal traces as part of the training data, and a different remaining 20% of the normal traces as part of the testing data. The results of the 10 runs were averaged for each experiment.

Table 3 shows the (average) anomaly detection results of these experiments alongside the results from (Forrest et al. 1996). Here the columns A, B, C, and D are the percentages of abnormal regions (measured by the respective experiment) of the traces. The column Forrest et al. is the percentage of the abnormal sequences of the traces

From Table 3, we can see that in general, intrusion traces generate much larger percentages of abnormal regions than the normal traces. We call these measured percentages the "scores" of the traces. In order to establish a threshold score for identifying intrusion traces, it is desirable that there is a sufficiently large gap between the score of the normal *sendmail* traces and the low-end scores of the intrusion traces. We observe that such gap in A (experiments with sequence length 11) is larger than the gap in B (experiments with sequence length 7). Further, the gaps in A and B are larger than the gaps in columns C and D. The RIPPER rules from experiments A and B describe the patterns of the normal sequences. Here the results show that these rules can be used to identify the intrusion traces, including those not seen in the training data, namely, the *decode* traces, the *sm565a* and *sm5x* traces. This confirms our earlier conjecture that rules for normal patterns can be used for anomaly detection. The RIPPER rules from experiments C and D specify the patterns of abnormal sequences in the intrusion traces included in the training data. The results indicate that these rules are very capable of detecting the intrusion traces of the "known" types (those seen in the training data), namely, the *sscp-3* trace, the *syslog-remote-2* trace and the *syslog-local-2* trace. But comparing with the rules from A and B, the rules in C and D perform poorly on intrusion traces of "unknown" types. This confirms our conjecture that rules for abnormal patterns are good for misuse intrusion detection, but may not be as effective in detecting future ("unknown") intrusions.

The results from (Forrest et al. 1996) showed that their method required a very low threshold in order to correctly detect the *decode* and *sm565a* intrusions. While the results here show that our approach generates much "stronger signals" of anomalies from the intrusion traces, it should be noted that the method in (Forrest et al. 1996)

did not use any abnormal sequence from the intrusion traces in training.

The RIPPER rule sets from our experiments have 200 to 280 rules. Each rule typically has 2 or 3 (positional) attribute tests. Therefore on average, 100 to 140 rules are checked before a sequence is classified as "normal" or "abnormal". In (Forrest et al. 1996), a sequence is looked up (matched) in a normal (sequences) database with ~1,500 entries in order to classify the sequence. Clearly, the RIPPER rule sets here have much smaller sizes (storage requirements). We speculate that these RIPPER rule sets are more efficient (faster) in classifying sequences. We plan to study further on how to optimize (compile) the rule sets to boost the efficiencies.

Learning to Predict System Calls

In the second set of experiments, we aim to learn the (normal) correlation among system calls: the *n*th system calls or the middle system calls in (normal) sequences of length *n*.

The learning tasks were formulated as the following:

- Each record has *n*-1 positional attributes, p_1, p_2, ..., p_{n-1}; plus a class label, the system call of the *n*th position or the middle position.
- The training data is composed of (normal) sequences taken from 80% of the normal *sendmail* traces.
- The testing data is the traces not included in the training data, namely, the remaining 20% of the normal *sendmail* traces and all the intrusion traces.

RIPPER outputs rules in the following form:

38 :- p_3=40, p_4=4.
[meaning: if p_3 is 40 (*lstat*) and p_4 is 4 (*write*), then the 7th system call is 38 (*stat*).]

102 :- p_1=106, p_4=101.
[meaing: if p_1 is 106 (*sigblock*) and p_4 is 101 (*bind*), then the 7th system call is 102 (*setsockopt*).]

...

5:- true.
[meaning: if none of the above, then the 7th system calls is 5 (*open*).]

Each of these RIPPER rules has some "confidence" information: the number of matched examples (records that conform to the rule) and the number of unmatched examples (records that are in conflict with the rule) in the training data. For example, the rule for "*38 (stat)*" covers 12 matched examples and 0 unmatched examples. We measure the confidence value of a rule as the number of matched examples divided by the sum of matched and unmatched examples. These rules can be used to analyze a trace by examining each sequence of the trace. If a violation occurs (the actual system call is not the same as predicted by the rule), the "score" of the trace is incremented by 100 times the confidence of the violated rule. For example, if a sequence in the trace has p_1=*40* and p_4=*4*, but p_7=*44* instead of *38*, the score of the trace is incremented by 100. The average score (by the total number of sequences) of the trace is then used to decide whether an intrusion has occurred.

Table 4 shows the results of the following experiments:

1. Experiment A: predict the 11th system call.
2. Experiment B: predict the middle system call in a sequence of length 11.
3. Experiment C: predict the 7th system call.
4. Experiment D: predict the middle system call in a sequence of length 7.

Each experiment was repeated 10 times, each using a different subset (80%) of the normal traces as part of the training data, and a different remaining 20% of the normal traces as part of the testing data. The results of the 10 runs were averaged for each experiment. In Table 4, the columns A, B, C, and D are the scores of the traces measured in the respective experiment.

Traces	A	B	C	D
sscp-1	24.1	14.3	24.7	13.5
sscp-2	23.5	13.9	24.4	13.6
sscp-3	23.5	13.9	24.4	13.6
syslog-remote-1	19.3	13.9	24.0	11.5
syslog-remote-2	15.9	10.9	23.0	8.4
syslog-local-1	13.4	7.2	19.0	6.1
syslog-local-2	15.2	9.0	20.2	8.0
decode-1	9.4	2.4	11.3	3.9
decode-2	9.6	2.8	11.5	4.2
sm565a	14.4	9.4	20.6	8.1
sm5x	17.2	10.1	18.0	8.2
*sendmail	5.7	1.2	12.6	0.6

Table 4. Detecting Anomalies using Predicted System Calls. Columns A, B, C, and D are the scores of the traces measured in the respective experiment. *sendmail* is the 20% normal traces not used in the training data. None of the intrusion traces was used in training.

We can see from Table 4 that the RIPPER rules from experiments A and D are very effective because the gaps between the score of normal *sendmail* and the low-end scores of intrusion traces are large. However, the rules from B and C perform poorly. In fact, the results of the 10 runs from both experiments B and C showed unusually large variations. Since B predicts the middle system call of a sequence of length 11 and C predicts the 7th system

call, we reason that the training data (the normal traces) has no stable patterns for the 6th or 7th position in system call sequences.

The rule sets from experiments A and D here have comparable sizes as well as performance (in terms of effectiveness) in detecting anomalies as the rule sets (for normal sequence patterns) from experiments A and B described in the previous section. Further analysis show that the intrusion traces have large numbers of consecutive violations of the rules that have confidence values as 1 (those that have no unmatched examples), whereas the normal traces have nearly none of such consecutive violations. This measurement of consecutive rule violations is similar to the post-processing scheme of counting abnormal "regions", and can be used as an alternative to detect anomalies. An important side effect of using the system call prediction rules for analyzing intrusion traces is that these rules can also record where and which system calls deviate from the normal correlation. System administrators can possibly use this information to learn the nature of the intrusions.

Discussion and Future Work

Our experiments show that the normal behavior of a program needs to be established in order to detect anomalous usage. This confirms the results of other related work in anomaly detection. The weakness of the model in (Forrest et al. 1996) may be that the recorded (rote learned) normal sequence database may be too specific. Their approach essentially requires the construction of a (nearly) complete normal database. Here we show that RIPPER was able to generalize the system call sequence information to a set of rules. We demonstrated that these rules, learned from a subset (80%) of the normal traces, were able to identify unseen intrusion traces as well as normal traces.

In order for our approaches or the methods in (Forrest et al. 1996) to be used for real-time intrusion detection, a model (a rule set or normal database) for each privileged program to be guarded needs to be built and maintained. Therefore the sizes and complexities of the models are critical to the performance of the detection system. The rule sets from our experiments are all very small and concise. Therefore our approach can potentially be incorporated into real-time systems.

The ultimate goal of an intrusion detection system is to have the capabilities to predict that an intrusion is about to occur and to detect that it is unfolding. Our approaches and the methods of (Forrest et al. 1996) can only detect that an intrusion has occurred after analyzing its trace. More information about program execution is needed in order to detect an ongoing intrusion. For example, in addition to the "action" information (e.g., the system call), the "subject" (the user that issues the action) and "object" (the resources touched by the action) can be included in building the models of normal program execution as well. 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 program audit trails that have 3 streams: subject, action and object.

We will also extend our experiments on predicting system calls to use some "ontology" information about system calls. In STAT (Ilgun et al. 1995), Unix system calls were categorized according to 10 action types, i.e., *Read*, *Write*, *Create*, etc., in the audit record preprocessor. We believe that using a similar grouping on the system calls, the resultant RIPPER rule sets will be more general and concise. What needs to be investigated is their accuracy in detecting intrusions.

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 with 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, 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, 120-128. IEEE Computer Society Press, Los Alamitos, CA.

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, 346-354.

Koral Ilgun, Richard A. Kemmerer and Phillip A. Porras. 1995. State Transition Analysis: A Rule-Based Intrusion Detection Approach. *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, 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.