Using Program Behavior Profiles for Intrusion Detection*

Anup K. Ghosh, Aaron Schwartzbard, & Michael Schatz
Reliable Software Technologies Corporation
21515 Ridgetop Circle, #250, Sterling, VA 20166
phone: (703) 404-9293, fax: (703) 404-9295
POC email: aghosh@rstcorp.com
www.rstcorp.com

Abstract

Intrusion detection and response has traditionally been performed at the network and host levels. That is, intrusion monitors will typically analyze network packet logs or host machine audit logs for signs of intrusion activity. More often than not, commercial off the shelf (COTS) intrusion detection tools use "fingerprints" of known intrusions to detect their presence in these audit trails. Both these approaches employed by most state-of-the-practice tools have their drawbacks. In this paper, we describe a method for program-based intrusion detection that is aimed at detecting novel attacks against systems.

1 Introduction

The field of intrusion detection is now beginning to reach commercial success as evidenced by the plethora of commercial intrusion detection tools on the market. One contributing factor to the success of intrusion detection tools is the failure of firewalls to prevent many security intrusions in practice. Firewalls are vulnerable to errors in configuration, ambiguous security policies, data-driven attacks through allowed network services, and insider attacks. One of the main problems with firewalls is not the lack of good technology, but rather the false sense of security it engenders in organizations that erect them. Once installed, individual host security is often relaxed and successful intrusions exploit the web of trust between computers to gain increasing levels of privilege.

Intrusion detection software can detect many of the intrusions that slip through or around firewalls. Defense-in-depth is the most effective strategy for securing systems today rather than single point solutions. While neither firewalls nor intrusion detection software is a silver bullet solution to security, together they provide a level of defense-in-depth that by themselves neither could provide.

Defense-in-depth can also be applied to intrusion detection system themselves. For example, maintaining multiple, diverse intrusion detection systems is often more effective than employing a single intrusion detection system.

Intrusion detection techniques are generally classified into one of two approaches: misuse detection and anomaly detection. Misuse detection methods attempt to model attacks on a system as specific patterns, then systematically scan the system for occurrences of these patterns

*This work is sponsored under Defense Advanced Research Projects Agency (DARPA) Contract DAAH01-98-C-R145. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

[4, 5, 6, 9, 12, 13]. This process involves a specific encoding of previous behaviors and actions that were deemed intrusive or malicious. Anomaly detection assumes that intrusions are highly correlated to abnormal behavior exhibited by either a user or network traffic. The main advantage of anomaly detection over misuse detection approaches is that novel attacks against systems can be detected by noting abnormal behavior. As a result, anomaly detection approaches are particularly popular among researchers [1, 9, 10, 8, 11].

One of the key problems with commercial intrusion detection tools is that most rely on misuse detection or signature analysis to detect intrusion attempts. Because these techniques attempt to match monitored data with patterns stored in databases, attacks that do not conform with known attack patterns will not be detected. In fact, even slight variants of known attacks often escape detection using signature analysis. The approach is limited by the breadth and depth of the knowledge stored in the pattern matching database. Practical constraints on storage of attacks is one limiting factor. Even more limiting for preventing intrusions, however, is the fact that novel or unknown intrusions will not be detected. Because new vulnerabilities are found and exploited on a daily basis, the signature analysis approaches currently on the market are reactionary and behind the curve pushed by computer crackers.

Desiring the ability to detect novel, unknown attacks against systems, an anomaly detection approach is employed in this research. Unlike traditional anomaly detection approaches, this research applies anomaly detection at the system process level. Rather than profiling the normal behavior of users, the approach here is to profile the normal behavior of executing processes. Monitoring at the process level abstracts away users' idiosyncrasies such that abnormal process behavior can be detected irrespective of individual users' behavior. Having baselined the normal behavior of a process, deviations from the normal behavior can be used to detect attempted exploitations of the program. Other intrusion detection work has been performed on system processes [2, 1, 3, 7].

2 Profiling Program Behavior

In this research, we profile a program's behavior by studying the set of system calls made by the program. One of the benefits of performing intrusion detection at the program level over network or host-based intrusion detection is having the ability to monitor intrusive behavior at a very fine grain level of accuracy. Computer security intrusions most often occur when programs are misused. System calls are a key indicator of program misusage. Because programs are only able to manifest system privileges by making system calls, analyzing the system calls made by a program is a reasonable approach to detecting intrusions based on program behavior profiles.

Several auditing facilities for a program's system calls exist on Unix systems including: Solaris's Basic Security Module (BSM), `ptrace`, `strace`, `ltrace`, and Solaris `/proc` interface commands. For the purposes of this study, we used BSM audit data provided to us under the DARPA 1998 Intrusion Detection Evaluation program[1]. The data was collected over a period of weeks in a controlled environment on an internal Air Force Research Laboratory (AFRL) network and provided to us by MIT's Lincoln Laboratory, who analyzed the results from each participant's study.

The data produced by BSM is a compact binary representation of system calls made by a program to the Solaris kernel. A tool called `praudit` is distributed with Solaris to convert the BSM data to human readable ASCII form. Individual sessions are programmatically extracted from the `praudit` data and partitioned into the BSM events corresponding to a particular program. For instance, for a session labeled:

```
08:39:27_0.0.0.0_in.telnetd,
```

[1] See `www.ll.mit.edu/IST/ideval/index.html` for a summary of the program.

the following BSM event files are created corresponding to the programs that make the recorded system calls:

```
login pt_chmod sh cat mail quota tcsch
in.telnetd mail.local sendmail.
```

Thus, a set of BSM events for each program in each session is created. The contents of anyone of these files may look like:

```
[pid 1387] execve;[pid 1387] open;[pid 1387] mmap;[pid 1387] open;
[pid 1387] mmap;[pid 1387] mmap;[pid 1387] munmap;[pid 1387] mmap;
[pid 1387] close;[pid 1387] open;[pid 1387] mmap;[pid 1387] mmap;[
pid 1387] munmap;[pid 1387] mmap;[pid 1387] close;[pid 1387] open;
[pid 1387] mmap;[pid 1387] mmap;[pid 1387] munmap;[pid 1387] mmap;
[pid 1387] mmap;[pid 1387] close;[pid 1387] open;[pid 1387] mmap;[
pid 1387] close;[pid 1387] open;[pid 1387] mmap;[pid 1387] mmap;[p
id 1387] munmap;[pid 1387] mmap;[pid 1387] close;[pid 1387] close;
[pid 1387] munmap;[pid 1387] open;[pid 1387] mmap;[pid 1387] memcn
tl;[pid 1387] munmap;[pid 1387] close;[pid 1387] close;[pid 1387]
close;[pid 1387] close;[pid 1387] exit;
```

where BSM events are semi-colon separated and the process ID of the process that creates the event is recorded. Though this file has only a single process (1387), it is possible for a given file to contain multiple, interleaved processes.

The session data provided by Lincoln Laboratory was labeled for training purposes such that intrusive sessions can be distinguished from normal sessions. Using these session labels, the intrusion sessions are separated from normal sessions. Because our approach is based on anomaly detection, we use only the normal sessions for training. For instance, to build a normal program behavior profile for the program `ls`, we first combine all the files named "ls" from all non-intrusive session directories into a single "ls" file.

A database is constructed for each program. The database is simply a table that consists of every unique N-gram of BSM events that occurred during any execution, as well as a frequency count for each N-gram. For instance, given the following snippet from a file of BSM events,

```
[pid 1] A;[pid 1] B;[pid 2] C;[pid 1] D;[pid 1] B;[pid 2] A;
[pid 1] D;[pid 1] B;[pid 2] E;[pid 2] B;[pid 2] D;,
```

and given an N-gram size of two, the following table would be created.

BSM N-grams	**Frequency**
A B	1
B D	3
D B	2
C A	1
A E	1
E B	1

Table 1: **Table of normal behavior profile for a program for an N-gram size of 2. Total number of unique N-grams is six, and the total number of N-grams is nine.**

The table shows the unique strings that are created from a program's BSM profile with their associated frequency. Each table characterizes the normal behavior of the program under non-intrusive usage. Tables such as this one are built programmatically for all programs under analysis. In this study, tables for 155 different UNIX programs were constructed. The tables are used by the intrusion detection algorithm to detect anomalous behavior.

3 A Simple Look Up Approach to Intrusion Detection

The algorithm we use for detecting intrusions is simple but effective. Similar to the approach of [2], we look up sequences of system calls captured during online usage in the tables of normal behavior built for each program. If the sequence of BSM events captured during online usage is not found in the database, then an anomaly counter is incremented. This technique is predicated on the ability to capture the normal behavior of a program in a database. If the normal behavior of a program is not adequately captured, then the false alarm rate is likely to be high. On the other hand, if the normal behavior profile built for a program includes intrusive behavior, then future instances of the intrusive behavior are likely to go undetected. Finally, it is important that the range of possible behavior patterns is much larger than the range of normal behavior patterns, because detection of intrusive events is only made possible when the intrusive behavior patterns are different from the normal behavior patterns. The more compact the representation of normal behavior and the larger the range of possible behavior, the better detection capacity this equality matching technique will have. The ratio of normal behavior patterns to possible behavior patterns provides an indication of the detection capacity of the technique. The smaller this ratio, the better the detection capacity.

Capturing system calls via the BSM provides a broad range of possible behavior. There are approximately 200 different BSM events that can be recorded. The N-grams described in Section 2 represent a sequence of N consecutive BSM events. Therefore a single N-gram can have 200^N possible combinations. In practice, however, a program will normally make only about 10 to 20 different system calls.

Interval	**Unit Contained**	**Threshold**
session	file	ST
file	window	FT
window	N-gram	WT
N-gram	BSM event	—

Table 2: **Using intervals of fixed sizes and thresholds to exploit temporal locality of attacks.**

The simple table look up approach employed here exploits temporal locality of anomalous sequences for detecting intrusions. That is, rather than averaging the number of anomalous sequences out of the total number of sequences, we employ thresholding functions in fixed-size intervals, where the fixed-size interval is much smaller than the total execution trace. A session is considered to be intrusive if it contains an anomalous execution of a program. A program is considered anomalous if a portion of its windows is anomalous. A window is considered anomalous if a portion of its N-grams is anomalous. An N-gram is considered anomalous if it cannot be found in the database corresponding to the program currently being checked.

Table 2 shows the fixed-size intervals, the units they contain, and their thresholds that are used to detect anomalous behavior. When the threshold for a fixed interval — such as a window of N-grams — is exceeded, the whole interval is marked as anomalous. Then a thresholding function

is applied at the next level up in abstraction using the marked anomalous interval to determine if an intrusion has occurred. For instance, if a window is marked as anomalous because its threshold, WT, for anomalous N-grams is exceeded, then this window is used as part of the count for file intervals to determine if the file threshold FT is exceeded. Ultimately, the session threshold, ST, must be exceeded for the session to be labeled as an intrusion. The thresholds are all configurable and the tuning of the thresholds will impact the performance of the system.

4 Results

The table lookup intrusion detection algorithm was evaluated by MIT's Lincoln Laboratory under the DARPA 1998 Intrusion Detection Evaluation program. Unlabeled sessions were sent by Lincoln Labs and processed by our intrusion detection algorithm. These sessions had an unspecified number of attacks of the following four types: denial of service (DoS), probe, user to root (u2r), and remote to local (r2l). Denial of service attacks deny access to resources for legitimate users. Probing attacks collect information about a host machine that may be used to stage attacks at a later time. A user to root attack is defined as an attack that elevates the privilege of a user with local account privileges. Remote to local attacks grant a remote user with no account privileges local user account privileges.

Attack	**Instances**	**Detections**	**% Detected**
DoS	8	3	37.5
probe	3	2	66.7
u2r	22	19	86.4
r2l	3	2	66.7
Total	36	26	72.2

Table 3: **Performance of table look up intrusion detection algorithm against different attack types.**

Table 3 shows the performance of the table look up intrusion detection algorithm for the different attack types for a threshold selected by the MIT Lincoln Laboratory. If the threshold for detection is set too low, then the false alarm rate will be low, but correct detection rate will be low, too. Similarly, a threshold set too high may end up detecting most intrusions, but suffer from a high false alarm rate. The results shown here appear to be a reasonable threshold for false alarms and detections.

False alarm rates are not shown for these different attacks because our algorithm will not label a particular attack — it only notes when an attack (any attack) is occurring. As a result, false positives cannot be tracked to particular attack types. However, for the threshold chosen, the total percentage of false positives for all the sessions evaluated was 2.1%. The table shows the intrusion detection algorithm detected only 3 out of 8 denial of service attacks. Four of these missed attacks were a "warez" attack that involved uploading files to an FTP server that permitted this action. Because no particular flaw in the FTP server was exploited and the FTP server configuration permitted this action, it is not surprising that our anomaly detection algorithm did not detect these four DoS attacks. The algorithm performed very well on the other classes of attacks. The attacks in Table 3 can be broken down into old and new attacks, as well as clear and stealthy attacks. The following tables show this breakdown.

Table 4 shows the performance of the algorithm for old and new attacks. An old attack is defined as an attack identical or similar to an attack that was present in the training data. A new attack

	OLD			NEW		
Attack	**Instances**	**Detections**	**% Detected**	**Instances**	**Detections**	**% Detected**
DoS	4	0	0.0	4	3	75.0
probe	0	0	0.0	3	2	66.7
u2r	16	13	81.2	6	6	100.0
r2l	1	1	100.0	2	1	50.0
Total	21	14	66.7	15	12	80.0

Table 4: **Performance of table look up intrusion detection algorithm for old and new attacks.**

is an attack that was not present in the training data. The results in this table show the power of the anomaly detection technique in detecting novel attacks against systems. The performance of the algorithm to novel attacks makes this technique viable in the ever-dynamic world of computer security intrusions.

	CLEAR			STEALTHY		
Attack	**Instances**	**Detections**	**% Detected**	**Instances**	**Detections**	**% Detections**
DoS	8	3	37.5	0	0	0.0
probe	3	2	66.7	0	0	0.0
u2r	13	11	84.6	9	8	88.9
r2l	3	2	66.7	0	0	0.0
Total	27	18	66.7	9	8	88.9

Table 5: **Performance of table look up intrusion detection algorithm for clear and stealthy attacks.**

Table 5 shows the performance of the algorithm in detecting clear versus stealthy attacks. Stealthy attacks attempt to hide perpetrator's actions from someone monitoring the system or from an intrusion detection system. For example, an attacker may edit the system log files after an attack to attempt to erase his or her tracks. The anomaly detection approach shows its strong ability to detect stealthy attacks.

In summary, the table look up intrusion detection approach performed well in all categories of attacks except denial of service. It correctly classified 72.2% of all attack sessions and 97.9% of all non-intrusion sessions. The low false positive rate (2.1%) is desirable for any intrusion detection approach. The intrusion detection algorithm's strengths are in detecting novel attacks against systems and in detecting stealthy attacks — both weaknesses of misuse detection or pattern matching intrusion detection approaches. Combining this approach with other intrusion detection approaches (such as network-based or misuse detection approaches) will provide a more robust intrusion detection defense that can detect old as well as novel attacks.

Acknowledgment

We are pleased to acknowledge Richard Lippmann, Marc Zissman, and Isaac Graf of MIT's Lincoln Laboratory for providing us the training and testing data, as well as evaluating our intrusion detection approach.

References

[1] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Security and Privacy*, 1996.

[2] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, October 1997.

[3] 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*, pages 120–128. IEEE, May 1996.

[4] T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, October 1991.

[5] K. Ilgun. Ustat: A real-time intrusion detection system for unix. Master's thesis, Computer Science Dept, UCSB, July 1992.

[6] S. Kumar and E.H. Spafford. A pattern matching model for misuse intrusion detection. The COAST Project, Purdue University, 1996.

[7] W. Lee, S. Stolfo, and P.K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management*, 1997.

[8] T.F. Lunt. Ides: an intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*, November 1990. Rome, Italy.

[9] T.F. Lunt. A survey of intrusion detection techniques. *Computers and Security*, 12:405–418, 1993.

[10] T.F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[11] T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannthan, C. Jalali, H.S. Javitz, A. Valdos, P.G. Neumann, and T.D. Garvey. A real-time intrusion-detection expert system (ides). Technical Report, Computer Science Laboratory, SRI Internationnal, February 1992.

[12] F. Monrose and A. Rubin. Authentication via keystroke dynamics. In *4th ACM Conference on Computer and Communications Security*, April 1997.

[13] P.A. Porras and R.A. Kemmerer. Penetration state transition analysis - a rule-based intrusion detection approach. In *Eighth Annual Computer Security Applications Conference*, pages 220–229. IEEE Computer Society Press, November 1992.

[14] T.H. Ptacek and T.N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., January 1998. Online. Available: http://www.securenetworks.com/papers/IDS.PDF.