Log Correlation for Intrusion Detection: A Proof of Concept*

Cristina Abad[†‡] *cabad@ncsa.uiuc.edu*
Jed Taylor[†] *jtaylr2@uiuc.edu*
Cigdem Sengul[†] *sengul@uiuc.edu*
William Yurcik[‡] *byurcik@ncsa.uiuc.edu*
Yuanyuan Zhou[†] *yyzhou@uiuc.edu*
Ken Rowe[§] *kenneth.e.rowe@saic.com*

[†] *Department of Computer Science, University of Illinois at Urbana-Champaign*
[‡] *National Center for Supercomputing Applications (NCSA)*
[§] *Science Applications International Corporation (SAIC)*

Abstract

Intrusion detection is an important part of networked-systems security protection. Although commercial products exist, finding intrusions has proven to be a difficult task with limitations under current techniques. Therefore, improved techniques are needed. We argue the need for correlating data among different logs to improve intrusion detection systems accuracy. We show how different attacks are reflected in different logs and argue that some attacks are not evident when a single log is analyzed. We present experimental results using anomaly detection for the virus Yaha. Through the use of data mining tools (RIPPER) and correlation among logs we improve the effectiveness of an intrusion detection system while reducing false positives.

1. Introduction

Information and resource protection is a primary concern of most organizations. To be able to secure their assets, organizations need effective intrusion detection techniques to respond to and recover from Internet attacks. An intrusion detection system (IDS) attempts to detect attacks by monitoring system or network behavior. While many existing IDSs require manual definitions of normal and abnormal behavior (intrusion signatures) [25], recent work has shown that it is possible to identify abnormalities automatically using machine learning or data mining techniques [6, 1, 2, 8, 14]. These works analyze network or system activity logs to generate models or rules, which the IDS can use to detect intrusions that can potentially compromise the system integrity or reliability.

However, most of the previous work on intrusion detection focuses on activities generated by a single source, resulting in many false positives and undetected intrusions. For example, some studies [14] detect intrusions by monitoring the network traffic such as tcpdump logs, whereas other studies [2, 8] monitor only system call logs. These studies have high false positive rates.

To address this problem requires correlating activities from all involved components and analyzing all logs together. This is because an intrusion typically leaves multiple signs of its presence, which to date has not been exploited by security professionals. The general idea is to take advantage of attack traces by correlating information found in multiple heterogeneous logs and thus enable IDSs to correctly identify more attacks while simultaneously reducing the number of false positives and providing a stronger validation that an attack has indeed occurred. Furthermore, we posit that there are many attacks that are not evident by analyzing a single log but may be exposed when correlating information in multi-logs.

We show in this paper that correlating log information is useful for improving both misuse detection and anomaly detection. To facilitate processing the millions of entries found in typical logs, data mining techniques are indeed useful. Specifically we use the data mining software tool RIPPER [3] and correlation to improve anomaly detection,

*This research is funded in part by a grant from the Office of Naval Research (ONR) within the National Center for Advanced Secure Systems Research (NCASSR) <www.ncassr.org>.

and present results that show that log correlation can improve current intrusion detection techniques.

The paper is structured as follows. Section 2 provides background and motivation for this work. The two complementary mapping approaches we use (top-down and bottom-up) are given in Section 3 and Sections 4-7 use these two approaches to examine attack signatures to better understand how they may be manifest in different audit logs. We show how multiple logs are affected by common attacks and give several specific examples of attacks and their attack traces in heterogeneous logs. In Sections 8 and 9 we outline a specific experimental procedure and present results for the case of anomaly detection for the virus Yaha, and show that by using log correlation and data mining techniques we are able to increase the effectiveness of an IDS by reducing the number of false positives. The areas of future work are listed in Section 10. Section 11 provides our summary conclusion.

2. Motivation and background

The challenges in securing a computer network can be viewed in three stages [27]:

- Prevention: Avoid intrusions if possible. Vulnerabilities should be identified and patched. It is important to use security measures such as user authentication, authorization and access control.
- Detection: Know as soon as possible when an intrusion occurs. Installing and using well configured anti-virus software, firewalls and IDSs helps early detection.
- Reaction: Respond to an intrusion. Problems should be fixed, intruders stopped and data restored.

Since it is unlikely that we will ever be able to prevent all intrusions, being able to detect an intrusion is fundamental. As long as traffic flow is allowed between an internal network and the Internet, the opportunity for an attacker or worm to penetrate will always exist. Despite the preventive steps taken, intrusions that involve new techniques usually succeed. During and after an attack, network and system logs are the primary sources of information for detection, response, and forensics, without these logs there is little chance of ever discovering what happened.

An IDS is designed to analyze computer and network activity to detect intrusion attempts as they occur. Intrusion detection systems can be categorized as follows [27]:

- *Network Intrusion Detection Systems* monitor packets on the network and attempt to discover if an an intruder is attempting to break into a system (or cause a denial of service attack). A typical example is a system that watches for a large number of TCP connection requests to many different ports on a target machine, thus discovering a TCP port scan.
- *System Integrity Verifiers* or *Host Intrusion Detection Systems* monitor system files to find if a change is made by an intruder, e.g. "Tripwire" [12, 13].
- *Log File Monitors* monitor log files generated by network services. A typical example would be a parser looking for patterns in HTTP server log files to discover attacks such as "phf"[1] [11] attack.
- *Deception Systems*, e.g., honeypots, contain pseudo-services that emulate systems and networks of systems including well-known holes in order to trap hackers.

The majority of commercial IDS solutions are network-based. One way to approach network intrusion detection is to detect statistical anomalies. The idea behind this approach is to measure a baseline status such as CPU utilization, disk activity, user logins, file activity and so on. Then the system can trigger an alert when there is a deviation from the baseline. However, such anomaly detection systems tend to produce frequent false positives since users and systems tend to deviate from normal over time. Therefore, the majority of the network intrusion detection products are based on signature recognition, i.e., they examine traffic looking for well-known attack patterns. However, signature recognition (or misuse detection) fails when an attack has many slightly different variations thus causing many false negatives.

A major shortcoming common to all security components (firewalls, IDSs, antivirus) is that individually they can be avoided, spoofed, or gamed. For example, IDS signatures or IDS placement within the network can be determined from probing over time and then the IDS can either be evaded by manipulating packets or gamed by intentionally triggering alarms (intentional false positives or squealing). Several successful attacks against existing IDSs are identified in [19, 21]. An emphasis on using data from multiple independent logs will make such evasion or squealing attacks more difficult.

Another shortcoming of current approaches in intrusion detection systems is that they only attempt to detect and prevent individual attacks and not coordinated distributed attacks. Much of the work centers around improving the ability of systems to detect attacks and the speed of the traffic that can be handled. Both signature-based and anomaly detection sensors are useful tools but they only provide a snapshot of the event in time after it has transpired. On the other hand clues as to what is happening in real-time on a computer network is available but spread out all over

[1] The phf CGI program contains a bug in the argument parsing procedure. By inserting new line characters in the script parameters it is possible to execute a command as the user running the web server.

the network. For instance, all modern operating systems can log system and security events. Servers keep extensive records of their operations. Many applications write errors, warnings, and failures to their own logs. Firewalls can track how many packets are dropped by different rules typically at the entry and exit points of the network. If all this information can be brought together it would enable more robust attack detection mechanisms.

An intrusion correlation system should analyze activity which might initially appear to be unrelated, in order to determine whether it is a part of an intrusion attempt and to characterize that pattern. Consider the case of an attacker who attempts to crack a password and moves from workstation to workstation in order to avoid detection. This method would not raise any alarms under normal circumstances but with correlation it is possible to see the big picture by receiving events from all locations and raise an alert.

Correlation makes detection more effective and provides information needed for future prevention and reaction. The primary goals of attack trace correlation are to reduce false alerts (i.e., false positives) and identify stealthy attacks (i.e., reduce false negatives). To achieve the aforementioned goals, most of the previous log correlation research focuses on collecting logs in the network to a central point and then using data reduction and correlation to detect intrusions. Correlating multiple logs enables an IDS to make a better decision of whether to trigger an alarm or not, if an anomaly is observed. neuSecure [17] is a commercial security event management solution that correlates files from firewalls, IDSs, computer systems and routers to provide a comprehensive view of a security events occurring throughout a system. netForensics [16] and Tivoli [9] are other commercial products available that provide data aggregation and simple correlation. Silvestro [24] in his Master's Thesis examines real-time correlation and implements a centralized correlation engine which is an expert system using data collection, aggregation and correlation rules.

We believe that correlating multiple logs in a single host is also important to increase the efficiency of detecting intrusions. We define employing log correlation in a single host as host-based correlation. By integrating different information from multiple logs, it is possible to reduce false alarms and increase detection rate.

Some related research exists in the area of correlating IDS alarms to improve the intrusion detection process [5, 18, 20]. Although the main goal and approach are similar to our work, these papers differ in that they depend solely on the alerts generated by the IDSs. This leads to solutions that are tailored for particular IDSs, and what is worse, they depend only on the limited view each IDS has of the network (i.e., they do not have access to data from logs not analyzed by the IDS).

3. Methodology

Our main goal is to show that log correlation can help improve intrusion detection systems. To achieve this goal, an analysis of how different attacks are recorded in different logs and how correlating these logs can help identify the attacks is needed. To accomplish this we use two complementary approaches to map attacks and the attack traces in logs: (1) "top-down" and (2) "bottom-up".

3.1. Top-down approach (attacks → logs)

In the top-down approach we take a known attack and analyze its behavior to infer which logs may contain attack traces. This approach reveals common behavior among classes of attacks. It is also possible to identify logs that are more likely to store useful information for entire classes of attacks. The major advantage of this approach is that it does not require detailed analysis of large log files as is common for most computer networks. We can perform theoretical analysis on a wide variety of attacks without implementation details (simulation models or actually performing attacks in a laboratory environment). As a result of this top-down approach Table 1 maps logs that are relevant to different classes of attacks.

For example an attacker performing a Winnuke attack (a DoS attack against the DNS system) may take the following steps to learn if a system is vulnerable [4]. First, use "nslookup" to locate the DNS server. Then, pings to check if the service is active. Finally, scans port 139 (NetBios) to learn if the Windows system is active. To detect this attack, we should examine the DNS, NetFlow, and syslog logs.

3.2. Bottom-up approach (logs → attacks)

In the bottom-up approach we try to gather relevant information from multiple logs to identify specific attack instances. Starting with simple events such as authentication failures and numerous ping messages, we try to discover potential attacks. Once an anomaly is detected in one of the logs, the next step is to examine other logs in the same time period. A decision tree can be used to determine which logs should be next examined. A major advantage of this approach is that new undocumented attacks can be discovered. Aggregation is useful in this approach to group and discard normal activity such that only suspicious activity is left for correlation.

4. Attacks and vulnerabilities

As a part of the top-down approach, we studied several attacks and vulnerabilities to determine how their effects can show up in different kinds of logs.

Table 1. Log correlation table

Attack	Log								
	Syslog	Firewall	Netflow	TCP	DNS	Auth	Web	Mail	FTP
Dictionary	×	×	×	×		×	×	×	×
FTP-Write	×			×		×			×
Imap	×	×	×	×				×	
Named	×		×		×				
Phf	×			×			×		
Sendmail	×	×	×	×	×	×		×	
Xsnoop	×		×						
Apache2	×	×	×	×			×		
Back	×			×			×		
Mailbomb	×	×	×	×				×	
SYN Flood	×	×	×	×	×				
Ping of Death		×	×	×					
Process Table		×	×	×				×	
Smurf			×	×					
Udpstorm			×	×	×				

The SANS website [23] lists the following top ten vulnerabilities in Unix:

1. Remote Procedure Calls
2. Apache Web Server
3. Secure Shell
4. Simple Network Management Protocol
5. File Transfer Protocol
6. R-Services and Trust Relationships
7. Line Printer Daemon
8. Sendmail
9. BIND/DNS
10. General Unix Authentication, accounts with no passwords or weak passwords

In his Master's Thesis, Kris Kendall [11] gives a description and signature of several attacks, which exploit the vulnerabilities cited above. Of the attacks listed in Kendall's thesis, we analyzed the following:

- **Remote to User Attacks** One in which the attacker, after exploiting some vulnerability, gains unauthorized local access to a machine over a network as a user of that machine. Examples: Dictionary, FTP-Write, Imap, Named, Phf, Sendmail, Xlock, Xsnoop.
- **Denial of Service Attacks** One in which the attacker overloads a computing or memory resource, or denies legitimate users access to a machine. Examples: Apache2, Back, Mailbomb, SYN Flood, Ping of Death, Process Table, Smurf, Udpstorm.

5. A top-down approach to log correlation

There are many independent logs that store information for their own purposes. They are sometimes not well-organized, and information overlap exists among the logs. We have taken the following logs into account in the process of top-down approach: NetFlow log, firewall log, syslog, TCPdump, DNS log, authentication log, web log, mail log, and FTP log.

Instead of concentrating on system vulnerabilities, each of which can be exploited in several different ways, we will analyze specific attacks (as listed in [11]) and how they affect the different logs. We believe that these attacks can be placed in one of two groups: denying access to others (either under strenuous load or from invalid queries) or granting access to oneself (usually for malicious purposes).

There are many attacks, at least one for every exploit/vulnerability. Table 1 shows where a sampling of attacks affect different logs. After analyzing how attacks affect each log, we are able to identify two important logs that need to be checked first: syslog and NetFlow. Analyzing these two logs and correlating the information found in them will help improve IDS performance. It needs to be noted that analyzing syslog is not trivial, as it contains information from several different system logs which is typically a large amount of data. Data mining techniques can be used to filter the important information in these logs.

6. An example mapping: Attack to log attack traces

As part of the top-down approach we show how a specific attack is reflected in different logs. Next is an example of the logged actions of a real attacker's attempt to use a system for malicious purposes [26].

1. DNS version query probe is used to determine if BIND is vulnerable. This action is logged in syslog.

2. System is attacked with a 'noop' attack. This can be seen in the TCP dump by looking at TCP packets with many noops, which usually indicate a buffer overflow attack.

3. Users are created to gain root access. The telnet log reveals that after obtaining root access of the system, the attacker creates two users with the same password, twin and hantu; the second with superuser access.

4. Intruder gains superuser access and now controls the system. Syslog shows sessions opened for the newly created users.

5. Attacker tries to use the compromised system for malicious purposes. The IDS running on the system (Snort [25]) logs the actions of the attacker:

   (a) Telnets to system using as "twin", then gains superuser access as "hantu".

   (b) Downloads (ftp) attack toolkit from another system.

   (c) Installs backdoor which allows unauthorized access to anyone with the TERM set to vt9111.

   (d) Covers moves by deleting affected logs.

   (e) Days later comes back and logs in using the backdoor.

   (f) Downloads and installs Trinoo [7] client.

   (g) Several attempts are made to use the system as a Trinoo DDoS attack against other systems.

Finally, the sysadmin disconnects the system so the attacker cannot use it for malicious purposes.

7. Bottom-up approach: A case of anomaly detection

As mentioned in Section 3, a bottom-up approach to finding attacks through correlation of data between logs requires an actual logging infrastructure in an attempt to identify attack. The problem with this approach is that each log may have millions of entries in it, and analyzing them is a computation-intensive task that may be intractable for even relatively small computer networks. Data mining techniques can be used to filter out the important data from different logs and to enable the identification of different attacks that may have occurred. In this section we use the bottom-up approach with a data mining tool (RIPPER [3]) to detect anomalies (in particular, the presence of the Yaha [22] virus).

We used the "predict the next system call" method combined with log correlation to improve virus detection. This method was briefly mentioned in the work of Lee and Stolfo [14, 15] but not explored.

There are currently several e-mail born viruses circulating around the Internet that have been difficult to eradicate, e.g., Yaha [22] the e-mail virus analyzed in our work. Yaha is first delivered to a user's mailbox; once the user opens the infected message, the virus searches the machine for e-mail addresses and sends off approximately 115 infected messages to the e-mail addresses that it finds. This intrusion takes place in the background and is unknown to the user, and the 115 messages do not show up in the users Sent items box.

Looking at the system call sequence, abnormal behavior may be observed which is not deemed statistically significant to signify an intrusion in a tuned IDS. However, if we examine the corresponding network traffic the traces show an unusual amount of network traffic corresponding to the system call sequence where the user opens the e-mail. Combining this information from the system call sequence and the network traffic, it is possible to identify intrusions that would normally not be identified using just one log.

Previous work in literature (Wenke Lee and Sal Stolfo [14, 15]) mine system calls or network data but do not correlate information between the two sets of data.

Lee et al. [14, 15] discuss two data mining classification models for anomaly detection. We use RIPPER [3] as the data mining tool for rule learning in two approaches:

1. The first approach is to feed training data to RIPPER with both "normal" and "abnormal" (system call or network) sequences. From that, RIPPER outputs rules which can be applied to predict whether a (system call or network) sequence is "normal" or "abnormal".

2. The second approach is to train RIPPER with "normal" traces only. Each output rule of RIPPER must have some confidence information: the number of matched examples and the number of unmatched examples in the training data. These rules are used to analyze new sequences.

The first step for both approaches is to train RIPPER with normal traces of both network traffic and system calls. For this we generated hours of normal e-mails/requests from the e-mail server. We captured the system calls along with the network traffic on a machine running Windows XP. The experiments were conducted on two interconnected workstations in a laboratory isolated from the Internet. The Windows XP machine had an e-mail client and logging tools installed and the Linux box served as an e-mail server. We used APISpy32 [10], with some modification, to intercept system calls. The logged system calls and network traffic on the client machine were then mined to determine if an intrusion had taken place.

Then second approach uses a system call sequence to predict the next system call. After processing the training data, RIPPER generates a series of rules. One such rule may be: if system call 1 is Open, then system call 7 should be Closed (with a confidence level of 0.38).

After RIPPER generates the rules, the confidence level of each rule is calculated as the number of matched examples divided by the sum of matched and unmatched examples. When mining the actual experimental data from both the network traffic log and system call log, the confidence

level of the violated rule is adjusted similar to Lee et al. [14]. The averaged score (by the total number of sequences) of the trace is used to decide whether there is an intrusion.

The main goal of our experiments was to test if this method may improve intrusion detection systems when the results from multiple logs are correlated (specifically traffic logs and system call logs). For example, if there is a slight anomaly in network traffic the confidence levels may change when mining the system calls. Unusual behavior found in a network traffic log may be an indication of an intrusion so the confidence level of certain rules for the system call log may be increased. Thus, we are able to identify intrusions with higher confidence levels that would normally not be identified by using a single log (traffic or system log).

8. Design and implementation

This section describes the different stages of design and implementation in detail. More specifically, we present the architecture and mining process, and how we correlate system calls with network data.

8.1. Architecture

Our setup consisted of 2 Dell Workstations with 2.56 GHz Pentium 4 processors and 512 MB RAM. Our architecture consists of logging utilities that monitor system activity from different vantage points. For example, APISpy32 [10] monitors the system from the kernel level by recording the system calls executed by the system. WinDump monitors the system from the network level. The rest of our architecture consists of several Perl scripts used in the correlation process.

8.2. Training

The training data (1.9 million data points) was generated by five users. Each spent approximately 4 hours recording "normal" and "abnormal" data. In order to capture "normal" behavior, users followed a written script which consisted of sending and receiving several e-mail messages. This included performing operations that each user normally does when using e-mail. We call each execution of the script a "trace". In order to capture "abnormal" behavior, the same script was used with the addition of an e-mail message infected with the Yaha32 [22] virus.

8.3. Formatting data

The next step was to format the data for RIPPER [3]. RIPPER expects a list of system calls followed by the value that it will predict. For example, if we decide that we are going to use five system calls to predict the sixth, then RIPPER expects the five system calls followed by the sixth that it will try to predict.

A sliding window was used to scan the normal traces and create a list of unique sequences of system calls. The size of the sliding window was determined by empirical experiments.

We tested several different sequence sizes (odd number sizes from 5-19). These sequences are constructed from a sliding window. For example, the first sequence would be system calls 1-5, while the second sequence would consist of system calls 2-6.

8.4. Generating rules

RIPPER processes the formatted sequences to generate a list of rules. Since there were almost two million data points, it took a long time to generate rules. The smaller sequence sizes took less time (size 5 took 15 hours). The larger ones took significantly longer (size 19 took five days).

Taking an idea from Lee and Stolfo's work [15], in order to get meaningful results for network traffic it is necessary to introduce some type of temporal information into the equation. Therefore, we decided to introduce the number of connections within the past 10 seconds as a data item. The rationale behind this decision is that a sudden spike in network traffic could be a good sign of an intrusion. In short, RIPPER used the characteristics from the network traffic connections in order to predict how many network connections have occurred in the past 10 seconds.

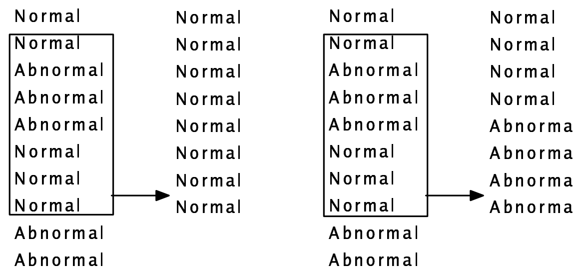
8.5. Applying rules

Approximately 20% of the captured data was saved to use for testing only; i.e., it was not used in training. The generated rules were applied to the testing data to determine how abnormal the traces were. RIPPER returned a list that consists of each predicted result (predicted based upon the rules) along with the actual value. It also gives the confidence level for the particular rule that was used in the prediction.

8.6. Post processing of system calls

A sliding window of size 13 was passed over the list of predicted values and actual values returned by RIPPER.[2] If the predicted value differs from the actual value, the penalty value is set to the confidence level of the rule that was broken. Then the penalty values are summed from the 13 spaces of the sliding window and if the value is greater than half (6.5), this "region" is considered as abnormal, otherwise it is considered as normal.

[2] Size 13 was selected for the post processing window based on testing.



(a) Threshold = 3.5 (b) Threshold = 2.5

Figure 1. Sliding window in the correlation process

The reason we chose to use the confidence level as a penalty value is to avoid rules having equal weight. For example, a rule that has a confidence level of 50% should not have the same penalty cost as a rule with confidence level of 99%. This differs from the work from Lee et al. [15]; they do not use the confidence level when calculating the penalty values when post processing.

Finally, the number of abnormal regions is added up and divided by the total number of regions. This value represents how abnormal a particular trace is.

8.7. Post processing of network traffic

The network traffic was processed in the same way as the system calls. The rules generated from the training data were used to predict the number of connections in the past 10 seconds.

The rules generated by RIPPER determine that the number of connections in the past 10 seconds should be 10. Next, we determined that if the value is actually between 0-20, we can still consider that normal. If there are 20-40 connections we say that the penalty is 1, and from 40-100 the penalty is 3. If there are 100 or above then we say that the penalty is 5.

Consider a trace that consists of values from 1pm to 2pm. If we see 50 connections in the past 10 seconds from 1pm to 1:10pm then we say that the penalty from 1pm to 1:10pm is 3, and so forth. These values were used in post processing.

8.8. Correlating system calls and network traffic

Figure 1 illustrates the correlation process. Figure 1(a) shows how the sliding window is used to detect if a region is abnormal. In Figure 1(a) the region is normal, since there are only 3 abnormals and the threshold is 3.5. In Figure 1(b) we lower the threshold by 1. Notice that the same region is now considered abnormal. Lowering the threshold is equivalent to increasing suspicion.

The results from the previous step were used in processing the system calls again, by following the same steps in Section 8.6 except that the time sequences that were determined were used along with their penalty values in correlating the data. The penalty value was used to adjust the threshold to determine if a region is “abnormal”.

Then the system calls were processed. When there was a sequence with a time value between 1pm and 1:10pm the threshold was lowered by the penalty value (3). This has the effect of increasing the suspicion level based on the data in a different log. After the correlation, it only took a 3.5 value to declare a region as abnormal as opposed to the original 6.5 value.

We also decided to increase the threshold during normal time periods. We expect this to help to eliminate false positives. For example, if network traffic is normal between 1:10pm and 1:30pm we can increase the threshold to detect an abnormal region by 1. So, it would take 7.5 to label a region as abnormal during that time period.

9. Results

The results obtained from the experiments are summarized in Tables 2 and 3. Table 2 shows the normal behavior traces and Table 3 shows the abnormal behavior traces. Traces t3, t7, t10, t14, t45, t23, t24, t51, t53, t33, t35, t40 are all normal behavior traces. While traces t100-t104 are all abnormal behavior traces. “Sequence size” is the number of system calls that we used to predict the next one. “No Corr.” is the percentage of abnormality when no correlation was done. “With Corr.” is the percentage of abnormality after the data from the two logs was correlated. “Diff.” is the change in abnormality after we introduce the correlation.

Our experiments produced interesting results. As is observable from the tables, the results show that correlation increases the accuracy of the intrusion detection system. After correlation, normal traces become more normal while abnormal traces become more abnormal (See Figure 2). We

Table 2. Normal behavior traces table

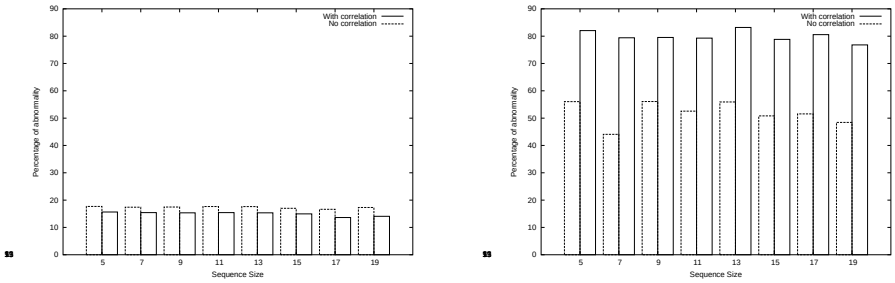
Sequence Size		No Corr.	With Corr.	Diff.
5				
	t3	8.19%	7.39%	-0.80%
	t7	16.31%	14.54%	-1.77%
	t10	31.16%	31.73%	0.57%
	t14	30.81%	29.86%	-0.95%
	t45	10.66%	10.21%	-0.45%
	t23	6.36%	6.04%	-0.32%
	t24	8.14%	7.59%	-0.55%
	t51	26.16%	25.32%	-0.84%
	t53	14.52%	13.87%	-0.65%
	t33	7.02%	6.73%	-0.29%
	t35	13.28%	13.52%	0.24%
	t40	17.89%	15.55%	-2.34%
				-0.68%
7				
	t3	7.87%	7.08%	-0.79%
	t7	15.97%	14.39%	-1.58%
	t10	30.85%	31.79%	0.94%
	t14	30.62%	29.73%	-0.89%
	t45	10.38%	9.94%	-0.44%
	t23	6.29%	5.96%	-0.33%
	t24	7.97%	7.42%	-0.55%
	t51	25.96%	25.19%	-0.77%
	t53	14.30%	13.74%	-0.56%
	t33	6.75%	6.59%	-0.16%
	t35	13.29%	13.29%	0.00%
	t40	17.41%	15.42%	-1.99%
				-0.59%
9				
	t3	8.26%	7.15%	-1.11%
	t7	16.70%	14.45%	-2.25%
	t10	30.38%	31.28%	0.90%
	t14	30.58%	29.74%	-0.84%
	t45	10.42%	9.92%	-0.50%
	t23	6.29%	5.95%	-0.34%
	t24	7.94%	7.42%	-0.52%
	t51	25.98%	25.17%	-0.81%
	t53	14.33%	13.77%	-0.56%
	t33	6.86%	6.60%	-0.26%
	t35	13.20%	13.12%	-0.08%
	t40	17.46%	15.34%	-2.12%
				-0.71%
11				
	t3	8.18%	7.11%	-1.07%
	t7	16.64%	14.38%	-2.26%
	t10	30.74%	31.65%	0.91%
	t14	30.59%	29.66%	-0.93%
	t45	10.39%	9.92%	-0.47%
	t23	6.33%	5.96%	-0.37%
	t24	8.03%	7.50%	-0.53%
	t51	25.99%	25.15%	-0.84%
	t53	14.45%	13.80%	-0.65%
	t33	6.93%	6.63%	-0.30%
	t35	13.35%	13.09%	-0.26%
	t40	17.55%	15.43%	-2.12%
				-0.74%

Sequence Size		No Corr.	With Corr.	Diff.
13				
	t3	7.85%	7.16%	-0.69%
	t7	16.00%	14.45%	-1.55%
	t10	30.54%	31.52%	0.98%
	t14	30.59%	29.66%	-0.93%
	t45	10.37%	9.91%	-0.46%
	t23	6.33%	5.95%	-0.38%
	t24	8.07%	7.53%	-0.54%
	t51	26.04%	25.17%	-0.87%
	t53	14.41%	13.72%	-0.69%
	t33	6.92%	6.64%	-0.28%
	t35	13.27%	13.02%	-0.25%
	t40	17.52%	15.34%	-2.18%
				-0.65%
15				
	t3	7.84%	7.17%	-0.67%
	t7	15.79%	14.48%	-1.31%
	t10	30.64%	31.51%	0.87%
	t14	30.48%	29.57%	-0.91%
	t45	10.34%	9.87%	-0.47%
	t23	6.32%	5.93%	-0.39%
	t24	7.96%	7.43%	-0.53%
	t51	25.96%	25.14%	-0.82%
	t53	14.35%	13.66%	-0.69%
	t33	6.99%	6.60%	-0.39%
	t35	13.29%	13.06%	-0.23%
	t40	17.01%	14.96%	-2.05%
				-0.63%
17				
	t3	7.83%	6.80%	-1.03%
	t7	15.92%	13.73%	-2.19%
	t10	30.85%	30.29%	-0.56%
	t14	30.38%	27.59%	-2.79%
	t45	10.32%	9.09%	-1.23%
	t23	6.66%	5.55%	-1.11%
	t24	7.97%	6.83%	-1.14%
	t51	25.91%	22.84%	-3.07%
	t53	14.27%	12.33%	-1.94%
	t33	6.86%	5.71%	-1.15%
	t35	13.23%	11.83%	-1.40%
	t40	16.64%	13.82%	-2.82%
				-1.70%
19				
	t3	7.97%	6.82%	-1.15%
	t7	15.91%	13.77%	-2.14%
	t10	31.03%	30.42%	-0.61%
	t14	30.60%	27.64%	-2.96%
	t45	10.40%	9.11%	-1.29%
	t23	6.38%	5.60%	-0.78%
	t24	7.91%	6.90%	-1.01%
	t51	25.96%	22.83%	-3.13%
	t53	14.31%	12.39%	-1.92%
	t33	6.96%	5.62%	-1.34%
	t35	13.32%	11.87%	-1.45%
	t40	17.30%	14.07%	-3.23%
				-1.75%

Table 3. Abnormal behavior traces table

Sequence Size		No Corr.	With Corr	Diff.
5				
	t100	39.28%	40.37%	1.09%
	t101	56.04%	82.04%	26.00%
	t102	27.21%	32.51%	5.30%
	t103	32.73%	43.23%	10.50%
	t104	26.33%	25.03%	-1.30%
				8.32%
7				
	t100	38.45%	39.66%	1.21%
	t101	44.06%	79.41%	35.35%
	t102	25.80%	31.52%	5.72%
	t103	30.39%	41.46%	11.07%
	t104	25.41%	24.52%	-0.89%
				10.49%
9				
	t100	38.96%	40.19%	1.23%
	t101	56.07%	79.53%	23.46%
	t102	27.26%	32.23%	4.97%
	t103	31.31%	40.35%	9.04%
	t104	26.37%	25.27%	-1.10%
				7.52%
11				
	t100	38.77%	40.22%	1.45%
	t101	52.56%	79.29%	26.73%
	t102	26.45%	32.01%	5.56%
	t103	32.09%	41.42%	9.33%
	t104	25.94%	24.89%	-1.05%
				8.40%

Sequence Size		No Corr.	With Corr.	Diff.
13				
	t100	39.03%	40.15%	1.12%
	t101	55.91%	83.18%	27.27%
	t102	26.74%	31.85%	5.11%
	t103	31.57%	41.03%	9.46%
	t104	26.15%	24.84%	-1.31%
				8.33%
15				
	t100	38.79%	40.02%	1.23%
	t101	50.83%	78.82%	27.99%
	t102	26.76%	31.89%	5.13%
	t103	31.38%	40.02%	8.64%
	t104	26.15%	24.84%	-1.31%
				8.34%
17				
	t100	38.64%	39.28%	0.64%
	t101	51.54%	80.56%	29.02%
	t102	26.61%	31.35%	4.74%
	t103	30.80%	39.27%	8.47%
	t104	26.10%	24..16%	-1.94%
				8.19%
19				
	t100	39.01%	39.13%	0.12%
	t101	48.44%	76.79%	28.35%
	t102	26.23%	30.08%	3.85%
	t103	29.48%	36.82%	7.34%
	t104	26.54%	24.25%	-2.29%
				7.47%



(a) A normal behavior trace

(b) An abnormal behavior trace

Figure 2. Percentage of abnormality in two traces, when different sequence sizes of system calls were used to predict the next one

believe that our data mining techniques could be applied to other IDSs to increase the accuracy of intrusion detection.

Although our results show an improvement in anomaly detection, we also determined that the "predict the next system call" method does not perform well for this particular problem. Normal traces had a high level of abnormal behavior to begin with, increasing the chance of false positives.

10. Future work

The results presented here are promising but more testing needs to be done to better understand the effects of multi-log correlation and the "predict the next system call method". In particular, we need to run more tests to determine the optimal sequence size and the optimal sliding window size for post processing. It would also be worthwhile to test different methods for correlation as well as test other data mining tools (other than RIPPER). It is also important to evaluate the use of different logs for the correlation process.

Nearly all offline-data mining projects can be enhanced by applying stream data mining techniques. Stream data mining involves data mining in real-time. We feel that our work could benefit from these techniques because once the rules are generated they could be applied on-the-fly.

11. Conclusions

Correlating logs to discover intrusion attacks is necessary to improve the effectiveness of the intrusion detection systems. By analyzing and correlating the information found in multiple logs, intrusion detection systems are able to improve the effectiveness of IDS alarms. However, correlating log information is not trivial. Audit logs have a large number of entries so data mining techniques are necessary to extract useful information from background noise.

Specifically in this paper we performed experiments to successfully correlate data from multiple logs in a laboratory setting for a particular case of anomaly detection (the Yaha virus). We report empirical results showing improved IDS accuracy by correlating network log and system call information. Not only were we able to make abnormal traces more abnormal, thus making intrusions easier to detect, but we were able to decrease false positives by making normal traces more normal.

12. Acknowledgments

This research is the result of two complementary student group projects (Spring 2003). The authors would like to acknowledge the hard work of student group members who performed many of the experimental procedures described in this paper (in alphabetical order): Andrew Bennett, Phil Cipriani, Valerie Kretschmer, Mathew Miller, Jungmin So, Yanli Tong, and Borshin Wang. Our work was influenced by a related project involving Elizabeth Partridge and Zhenmin Lee. We would also like to thank the following NCSA/UIUC colleagues who made significant indirect intellectual contributions to this paper: Jim Barlow, Ratna Bearavolu, Rafael Bonilla, Randy Butler, Jiawei Han, Kiran Lakkaraju, Yifan Li, Doru Marcusiu, Hrishi Raje, Bobby Rariden, Jeff Rosendale, and Xiaoxin Yin. Lastly, we thank the anonymous reviewers for their helpful feedback.

References

[1] D. Barbara, J. Couto, S. Jajodia, L. Popyack, and N. Wu. ADAM: Detecting intrusions by data mining. In *Proc. of the IEEE Workshop on Information Assurance and Security*, Jun. 2001.

[2] J. B. D. Cabrera, L. Lewis, and R. K. Mehra. Detection and classification of intrusions and faults using sequences of system calls. *ACM SIGMOD Record*, 30(4):25–34, Dec. 2001.

[3] W. Cohen. Fast effective rule induction. In *12th Intl. Conference on Machine Learning (ICML'95)*, 1995.

[4] F. Cuppens, F. Autrel, A. Miège, and S. Benferhat. Correlation in an intrusion detection process. In *SÉcurité des Communications sur Internet (SECI'02)*, Sep. 2002.

[5] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proc. of the 4th Intl. Symposium on Recent Advances in Intrusion Detection (RAID'2001)*, Oct. 2001.

[6] D. E. Denning. An intrusion-detection model. *IEEE Trans. on Software Engineering*, 13(2):222–232, Feb. 1987. Special Issue on Service Overlay Networks.

[7] D. Dittrich. The DoS project's "trinoo" distributed denial of service attack tool, Oct. 1999. <http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt>. (Mar. 15, 2003).

[8] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 120–128, 1996.

[9] IBM Tivoli intrusion manager, Jun. 2003. <http://www-3.ibm.com/software/tivoli/products/intrusion-mgr/>. (Sep. 1, 2003).

[10] Y. Kaplan. Api spying techniques for windows 9x, nt, and 2000, Apr. 2000. <http://www.internals.com/articles/apispy/apispy.htm>. (Feb. 12, 2003).

[11] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, MIT, 1998.

[12] G. H. Kim and E. H. Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. In *Proc. of the 3rd Annual System Administration, Networking and Security Conference (SANS III)*, pages 89–101, Apr. 1994.

[13] C. Kodology, R. Day, C. A. Christiansen, and J. Daly. Data and network integrity – technology to invoke trust in it – the tripwire solution, 2001. An IDC White Paper.

[14] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proc. of the 7th USENIX Security Symposium (SECURITY'98)*, Jan. 1998.

[15] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *Proc. of the IEEE Symposium on Security and Privacy*, May 1999.

[16] Netforensics: Security information management, 2003. <http://www.netforensics.com>. (Jun. 1, 2003).

[17] neuSECURE: Centralized security operations and threat management software, Aug. 2003. <http://www.guarded.net>. (Jun. 1, 2003).

[18] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proc. of the 9th ACM Conference on Computer & Communications Security*, pages 245–254, Nov. 2002.

[19] S. Patton, W. Yurcik, and D. Doss. An achilles' heel in signature-based ids: Squealing false positives in SNORT. In *Proc. of the 4th Intl. Symposium on Recent Advances in Intrusion Detection (RAID'2001)*, Oct. 2001.

[20] P. A. Porras, M. W. Fong, and A. Valdes. A mission-impact-based approach to infosec alarm correlation. In *Proc. of the 5th Intl. Symposium on Recent Advances in Intrusion Detection (RAID'2002)*, pages 95–114, Oct. 2002.

[21] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., 1998.

[22] L. Rohde. Yaha virus lingers into the new year, Jan. 2003. IDG News Service.

[23] Sans portal, May 2003. <http://www.sans.org/>. (Jun. 5, 2003).

[24] C. Silvestro. Intrusion detection systems and log correlation. Master's thesis, Cefriel: Consorzio per la Formazione e la Ricerca in Ingegneria dell'Informazione, Jun. 2002.

[25] Snort: The open source network intrusion detection system, Sep. 2003. <http://www.snort.org>. (Sep. 2, 2003).

[26] L. Spitzner, editor. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*, chapter 6. Addison-Wesley, Aug. 2001.

[27] J. Walker. Security event correlation: Where are we now, 2001. netIQ whitepaper. <http://www.netiq.com/products/sm/whitepapers.asp>. (Aug. 20, 2003).