Extracting attack manifestations to determine log data requirements for intrusion detection

Emilie Lundin Barse and Erland Jonsson
Department of Computer Engineering
Chalmers University of Technology
412 96 Göteborg, Sweden
{*emilie, erland.jonsson*}*@ce.chalmers.se*

Abstract

Log data adapted for intrusion detection is a little explored research issue despite its importance for successful and efficient detection of attacks and intrusions. This paper presents a starting point in the search for suitable log data by providing a framework for determining exactly which log data that can reveal a specific attack, i.e. the attack manifestations. An attack manifestation consists of the log entries added, changed or removed by the attack compared to normal behaviour. We demonstrate the use of the framework by studying attacks in different types of log data. This work provides a foundation for a fully automated attack analysis. It also provides some pointers for how to define a collection of log elements that are both sufficient and necessary for detection of a specific group of attacks. We believe that this will lead to a log data source that is especially adapted for intrusion detection purposes.

Keywords: *Intrusion detection, attack manifestations, log data, data collection*

1 Introduction

Intrusion detection is an increasingly vital part of general computer security, as computer systems today must cope with a wide variety of threats. A key issue is thus the presence of suitable log data that intrusion detection systems (IDS) can analyse for an accurate determination of the current system state. Unfortunately, the common adage of "garbage in, garbage out" also applies for intrusion detection. We need high-quality log data to provide high quality alerts, and to avoid false positives and false negatives. For that reason, it may come as a surprise to the uninitiated that even after 20 years since the birth of IDS, it is still not known what kind of log data that are needed to detect different types of intrusions and attacks. Furthermore, there are few guidelines to how such data should be collected and evaluated as well as how to systematically extract useful attack signatures from log data. Most logging mechanisms that exist today *have not* been created by computer security experts, and as a result they are not especially suitable for intrusion detection. As no better sources exist, they are still used for intrusion detection, and possibly compounding to the often-quoted failures of IDSs.

In this paper, we explore how logging can be ameliorated by investigating what *kind of* log data are needed. Our approach is empirical, and we start by considering the information found in current logging systems by developing a framework for determining log data requirements for different attacks. One central part of this framework is the extraction of *attack manifestations*. An attack manifestation is the log entries added, changed, or removed by an attack compared to corresponding normal behaviour. From these manifestations, we can choose combinations of log elements, i.e. log indicators, that can be used for detection and that could be part of a new intrusion detection log source. We demonstrate the use of the framework by studying different attacks in three types of log data. While the different steps of the framework should be further investigated and automated, this work provides a starting point in the development of future logging mechanisms.

The organisation of this paper is as follows. In Section 2, we discuss logging mechanisms used today, and related work. In Section 3, we describe the framework used for determining log data requirements of attacks. The analysis of three different attacks using the framework is presented in Section 4, and our findings are discussed in Section 5. Finally, we present our conclusions in Section 6.

2 Background

Today, the most popular log source used for intrusion detection is *network traffic*. The main reason for the extensive use of network traffic is that it is easily available and standardised. On the other hand, the network traffic does not

catch all events in the system. Also, it is difficult to keep up with the amount of data and it requires a great deal of computing power to parse the data part of the network packages in networks with much activity. Network traffic is used by e.g. Snort [Roe99].

Another log source is *system call logging*, which provides a more complete picture of the events in a single system but commonly adversely affects the performance of the target system. eXpert-BSM [LP01], to name an example, uses this type of log source.

The third type of logging used is *application-based logging*, in which the applications are instrumented to provide the log information. We count *syslog* to this category, as well as other specific application-integrated logging, e.g. in web servers and firewalls. Application-based logging can provide valuable information but both the syntax and semantics varies with the application in question, thus making this logging source difficult to use. For example, Swatch [HA93] can be instrumented for intrusion detection with syslog, and Almgren et al. [ADD00] have successfully used web server logs for application-based intrusion detection. Also, logging of user shell commands are counted to this category and has been used with success in [Max03].

Some IDSs use several log sources as input, though very few actually combine the information from the different sources to gain synergy effects in detection of single attacks. However, Vigna et al. [VRKK03] have created improved attack scenarios by including information from both web server logs, network traffic, and operating system events.

Keeping these log sources in mind, we now turn to what *kind of information* we would like to be able to extract from the log files. When faced with a possible attack, we might pose any of the following questions.

> *Did someone attempt to do something malicious? Did the malicious activity succeed? When did it happen? Who did it? From where was it done? What effect did the attack have on the system? Assuming a successful attack, what did then the attacker change in the system?*

Even if the syntax and semantics of different log sources varies, many provide a few similar fields that we can use to partly answer these questions. The *date & time stamp* can be used for determining when something happened, the *user name & id* can in the best case identify the attacker directly or at least show the subverted account, and the *location/host name/IP* is useful for determining the origin of the attack.[1] Apart from their direct usefulness, these fields play a vital role when correlating entries between log sources.

[1]Depending on the attack and the sophistication of the attacker, the source IP address may not be very useful. For example, hackers often leapfrog through hacked accounts and some DoS attacks use spoofed source addresses.

Information to answer the rest of the questions are not directly available, and depends on the type of attack we are dealing with. This is exactly the issue we are exploring in this article.

2.1 Related work

Axelsson et al. [ALGJ98] pointed out that much information found in log sources have little or no relevance for intrusion detection. They used only the `execve()` system call with arguments, and reported a detection of 21 out of 30 attacks with this lightweight logging method, which was better than for the "traditional" logging methods, including logging of system calls without arguments.

Abad et al. [ATS+03] study which attacks are covered by different logs, including Syslog, a firewall log, Netflow[2], TCP, DNS, Auth, Web, Mail, and FTP logs. They also study the usefulness of correlating information from these logs and come to the conclusion that better detection results can be obtained if more than one log source is used. However, they do not explain what they define as traces from attacks and how they extracted this information from the nine different log sources they have included in their study.

A taxonomy where attacks are classified by the complexity of their manifestations, i.e. the signatures they leave in log data, is created by Kumar [Kum95]. Some attacks manifest as single events, which is the simplest class of attacks. Others manifest as sequences or sets of events and are more complex to detect. The focus of Kumar's work is to classify the attacks by the computational effort required for detecting them which is useful in signature-based detection. While this is interesting work, it has a different focus than ours. First, Kumar does not describe how an attack is analysed to find the (least complex) signature that can be used for detection. We provide a methodology for analysing attacks. Second, Kumar uses only one manifestation for each attack. We believe that there are more than one way to detect the same attack, using different manifestations and different log data. Thus, our method can be used to find all the manifestations for an attack, giving a choice in which manifestation and what log data to use for detection.

The work closest to the method presented in this paper is a recent paper by Killourhy et al. [KMT04]. They extract attack manifestations (sequences of system calls without arguments) by comparing attack traces to normal traces for 25 attacks. The process used for extracting manifestations seems to be mainly manual. Their focus is to create a defence-centric taxonomy of attacks, which is achieved by studying how the attacks manifest as anomalies. The attacks are classified by the type of manifestations they produce, i.e. foreign symbols, minimal foreign sequences, dormant sequences, and non-anomalous sequences. They show

[2]NetFlow is a tool providing network statistics

that these classes corresponds well to how a sequence-based anomaly detector can detect the attacks. Their method seems to be rather close to our method. However, they focus on manifestations as system call sequences, while we apply our extraction method to several log types. Their goal is to classify attacks by their detectability using anomaly detection techniques and they classify each attack by the best, i.e. simplest, manifestation found. Our goal is to extract all log data from a data source that potentially can be useful for detection of a specific attack. Thus we focus on manifestations and log data useful for both anomaly and misuse detection, and we describe the extraction process in more detail.

3 Determining log data requirements

In this section, we present a framework for determining log data requirements for attack detection. The framework includes a number of steps where the attack is analysed, log data from the attack are compared to normal log data, and combinations of log elements that can be used for attack detection are determined. We do not provide full solutions to how each step is best implemented, but provide an outline and suggest some methods that can be used to analyse attacks and their appearance in log files. To summarise the logging, a program (or system) generates a certain number of log entries in different log sources when run. An attack affecting a specific program may *add*, *replace* or *remove* log entries that are normally produced from innocuous use of the program. These *changed* log entries are then used to flag malicious behaviour by an intrusion detection system. Our framework describes an outline of how to *extract* these changed log entries, *evaluate* their quality, and then using them to *identify* the log elements that can be used for detecting the attack. The methods are based on *controlled logging* of innocuous and malicious behaviour, including failed attack attempts. In the next section we introduce some terminology necessary to understand the framework, before we present it in Section 3.2.

3.1 Definitions

We define a **log entry** to be one line or post in a log file, e.g. one network packet in a `tcpdump` log file, or one system call in an `strace` log file. Each *log entry* consists of a number of **log elements**, i.e. data fields such as time, destination port, or TCP flags from a `tcpdump` log entry.

Events are generated in a computer system directly in response to *user actions*, or more indirectly as internal *system reactions* to the user action, or as a *system response* to the user. An *event* that is part of an attack is referred to as an **attack event**.

The *events* are grouped into **meta-events**, where one meta-event corresponds to a specific *user action*, followed by a number of *system reactions*, and the final *system response* to this user action. A complete attack consists of a sequence of *meta-events*. A *meta-event* generates a sequence of log entries in a specific log file, and these log entries are referred to as an **event trace**. In the same way, an **attack event trace** is the sequence of log entries generated by an **attack meta-event**.

Attack event traces are compared to traces of corresponding normal *meta-events* to find the sequences that are added, changed or removed by the attack. These sequences are called **attack manifestations**. Each *attack meta-event* can generate zero, one or more *attack manifestations*.

The collection of *attack manifestations* from a specific attack forms a subset of log entries. This subset is vital when one tries to detect the attack, because it contains the (only) information that distinguishes malicious from benign system activity. Under some circumstances, also the context surrounding the attack manifestation is important, and the subset is correspondingly expanded. Needless to say, the quality of the *attack manifestations* varies. For those reasons, we define an **attack indicator** to be a set of log elements from log entries of a single *attack manifestation* that (together with context information) can help distinguish an *attack event* from other events. Several attack indicators can be created from one attack manifestation.

3.2 Extraction framework

First, we present a summary of the log data extraction framework, and then a more detailed description of each step. The first three steps are preparatory, the actual logging and the log file comparison starts in step four, and the last three steps involve the analysis of the extracted log entries.

1. Enumerate user actions, and define the **attack meta-events** by analysing the attack.
2. Determine **normal events** to which the attack events can be compared.
3. **Classify** the attack meta-events.
4. Extract **event traces** by logging successful attack events, and the corresponding normal and failed attack events.
5. Extract **attack manifestations** by comparing traces.
6. **Classify** the attack manifestations.
7. Create **attack indicators** by using information from the attack manifestations.
8. Define the **log data requirements** of the attack by studying the attack indicators.

In step one, we enumerate (e.g. by running the attack) the *user actions* involved in the different stages of the attack. For each user action, a *meta-event* is created by finding the expected system response and system reactions.

In the second step, we analyse the attack meta-events to find suitable normal events that can be used for comparison

in step 5. This is a critical step, and we should collect events that correspond to the program being used in a valid way as well as failed malicious attempts.

In the third step, the attack meta-events are then *classified* to determine their importance. Meta-events that are not considered important can be excluded from the logging procedure.

Step four is the start of the actual execution of the attack events and the logging. It is advantageous to log each meta-event separately, if at all possible. That is, each user action is performed in sequence and the meta event is collected before the start of the next user action.

In the fifth step, we contrast the traces of successful attack event with traces of normal behaviour. These resulting *attack manifestations* show what the execution of the attack adds or removes in the log files, and from these log entries we can create attack indicators useful for detection. Figure 1 shows how the event traces are compared to extract attack attempt and attack success manifestations. The traces of failed attack events are also compared to normal behaviour traces, which gives a new set of attack manifestations. Those manifestations that appear in both comparisons described above indicate that an attack attempt is going on. Those manifestations that differ between the comparisons of successful attack to normal behaviour and failed attack to normal behaviour indicate the result of the attack. The

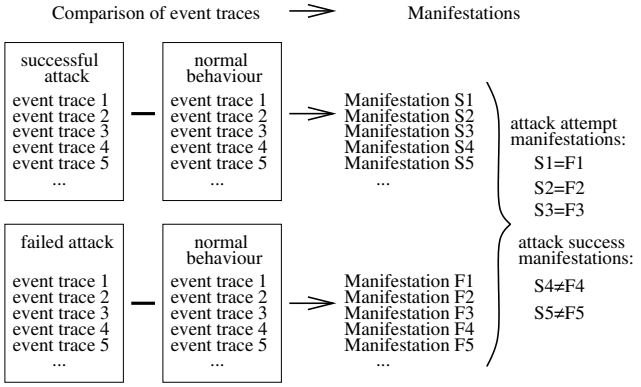

Figure 1. Manifestation extraction

traces are preferably compared using automated techniques. Methods for this comparison and extraction of attack manifestations in different types of logs are discussed in Section 3.3.

Step six contains the analysis and *classification* of the attack manifestations according to a number of criteria. Each manifestation has a number of properties that makes it more or less useful for the detection of the attack. For example, some are unique identifiers of the attack, and some are only supportive of an attack hypothesis; some can be logged with small resources, while other require resource demanding logging; some come from necessary attack events, and some may not be present in another attack script version. The event and manifestation categorisation is discussed in Section 3.4.

The *attack indicators* are created from the attack manifestations in step seven. The attack indicators consist of the minimal unique parts of the attack manifestations, which can be distinguished from the rest of the attack and normal event traces. This includes parts of single log entries, sequences of parts of entries, or counts of entries. In the system call logging case, the candidates for attack indicators are unique system calls, unique system call sequences, unique system call arguments or return values. Further discussion of creation of attack indicators is found in Section 3.5.

The last step of the method is to define the *log data requirements* of the created attack indicators. This is done by summarising the log elements used in the indicators.

3.3 Comparison of event traces

Comparing two log traces from an attack event and a normal event requires support from automated tools. Often many log entries are produced by a single event and these do not always end up in exactly the same order. The methods for comparing traces and extracting manifestations for the three different log types we have used for the attack evaluation are presented here.

System call logs. One way to compare *system call logs* is to use a method based on creating *diff files* similar to those created by the UNIX "diff" command. Here, the diff command matches text strings in the two input files and produces an output file with the lines that are added, removed or changed in the first input file compared to the second input file. The diff command can not be used directly because the arguments to some system calls differ almost every time they are used. Thus, we automatically remove irrelevant information, i.e. mainly process IDs, time stamps, and file descriptors, before using the traditional UNIX diff command. We then restore the removed log data fields from the original system call logs. From these diff files it was possible to manually extract indicators without much effort and knowledge about the function of the traced program. Manifestations, such as certain system calls only appearing in one of the files, differences in frequency of certain system calls, sequences of system calls that differ, and system call arguments that differ, could be seen rather easily. For this method to work well, one process at a time should be analysed, and preferably the two traces should be from the same program version.

Also, we supported the comparisons with system call frequency analysis and searches for strings that appear in the attack script and user names involved in the attack for extracting the attack manifestations. Counting the use of each

system call in the event traces clearly distinguishes calls that appear in one trace but not in the other.

Network traffic logs. Diff files similar to the ones we describe for the system call logs were created for the comparison of the network traffic logs. The elements in the network packet logs that were selected for comparison was source IP, destination IP, transport protocol, application protocol port, and data length.

We also used some visualisation methods to compare two logs and extract the differences. Packet types, packet sizes and the use of different protocol flags were useful to plot, to get a conception about similarities and differences in two packet streams. String searches were useful for finding packets containing special strings or file names that were connected to the attack.

Application logs are often designed to be manually readable and user actions do not generate a large amount of data in these logs. We have only used manual comparison for these logs in the attack evaluations in this paper.

3.4 Categorisation of events, manifestations, and indicators

The reason for classifying attack events and their manifestations is that it helps us choose which manifestations to use when creating attack indicators. The most useful manifestations are strong indicators of events that are malicious, always part of the attack and can only be performed in one way.

The reason for classifying attack indicators is that it helps us choose the best set of indicators, and thus also log elements, for detecting a group of attacks. This can form the basis for creation of a new "slimmed" log data source containing only necessary information. Another reason for classifying attack indicators is that it helps an IDS designer to choose the best set of indicators for detecting a specific attack. What the best set of indicators is, depends for example on the possibility to use different kinds of logging in the system and the processing power of the IDS. Also, the indicators in the set should complement each other, for example by including one that is useful for detecting the attack attempt, and another indicating the result of the attack.

Another goal of the classification is that it should be useful for quickly determining the (theoretical) detection capability of an IDS, for a specific set of attacks and for a specific set of input log data.

Below we suggest categorisations that can be used for evaluating the events, manifestations, and indicators. These are used in the presentation of our results in Section 4.

The Attack Phases are divided into three parts: *preparatory events* (P1), *attack execution events* (P2), and *compromised-system-use events* (P3).

The Attack Meta-Event Necessity Categories are divided into two levels: *necessary attack events* (N1) and *optional attack events* (N2). An attack event is classified as *necessary* if it is necessary for the attack to be successful. Obviously, indicators built of necessary attack events are more useful than those from optional attack events. The latter can easily be defeated by attack variations.

The Attack Meta-Event Variability Categories are divided into two levels: *invariable events* (V1) and *variable events* (V2). Even if an attack event in itself is N1, the particular methods used to perform the event may be exchangeable, resulting in a different attack trace.

The Attack Meta-Event Maliciousness Categories are divided into three parts: *violation* (M1), *indirect violation* (M2), and *benign events* (M3).

The Attack Manifestation Strength Categories are divided into two parts: *strong manifestations* (S1) and *weak manifestations* (S2), depending on how much information the manifestation reveals about the underlying event. A strong manifestation of a meta-event representing a violation is of course more useful for detection than a weak manifestation of a meta-event representing a benign event. A meta-event classified as benign (M3) can generate strong manifestations, but these only show that a specific (normal) event has taken place and can not be used for detection by themselves.

The Attack Indicator Application Categories are divided into five parts: *indicator for identifying the attack* (U1), *indicator for detecting illicit behaviour* (U2), *indicator for detecting attack result* (U3), *indicator for detecting service in vulnerable mode* (U4), and *indicator for detecting use of the vulnerable service* (U5).

The Attack Indicator Complexity Categories are divided into four parts: *single log entry* (C1), *sequence of log entries* (C2), *count of log entries* (C3), and *log entries requiring parsing to assemble event* (C4), related to the computing and memory resources needed to use the indicator in question.

3.5 Extraction of log indicators

Indicators are created from the manifestations by analysis of the included log entries. The goal of the analysis is to find the unique minimal parts of the manifestation that could be used for detection. For example, in the system call logs manifestations we analyse each system call to determine if it is unique, i.e. a *foreign symbol* according to Killourhy et al. [KMT04]. We also study the arguments and return values to see if they are unique. Sequences are studied to find out if they are unique. These sequences can also include the log entries surrounding the manifestation, and corresponds to the *minimal foreign sequences* used in Killourhy et al. [KMT04]. The last type of analysis is the frequency analysis of single system calls and sequences.

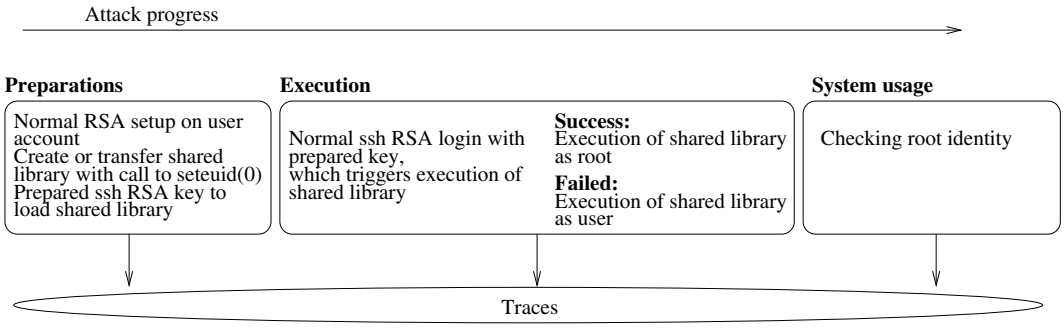

Figure 2. OpenSSH attack phases

3.6 Discussion of the framework

The goal of this work is to be of help in the search for suitable log data that can be included in a new log source for intrusion detection. However, the framework presented in this paper is still only an outline of how the log data extraction procedure can be done. We have suggested some partly automated methods for extracting manifestations and for creation of log indicators that we use for the attack analysis. We have also suggested categories for classification and evaluation of attack events, manifestations, and indicators. A complete methodology for extraction of log data requires further automation of each step and further investigation of how each step is best implemented. These issues are the subject of ongoing work.

We still believe that a new log source can be created from the lists of log data that are extracted in the attack analysis process described in this paper. By analysing attacks and selecting log indicators for which the log elements are common for a group of attacks, we can minimise the log data needs for that group of attacks. Another possible use of this framework is to help developers of intrusion detection signatures to find and choose suitable attack signatures.

4 Attack analysis

In this section, we describe experiments that we have carried out. We used the proposed framework and the suggested methods for analysing attacks to determine their log data requirements. We chose three different attacks and three different log sources for the first set of experiments. We tried to choose attacks that are different in their log data demands. Since no classification exists that is based on log data needs, the attacks were chosen on other characteristics that seemed relevant. One is a denial of service attack against a network protocol (Neptune SYN flood - CERT CA-96.21), one is a buffer overflow which affects the execution flow in the operating system (Tcpdump attack - bugtraq ID 1870, CVE-2000-1026), and one is a exploit of a privilege checking flaw (OpenSSH - bugtraq ID 1334, CVE-2000-0525). Three of the main types of audit sources are studied in the experiments in this paper. These are network traffic logged by *tcpdump*; system calls logged by *strace* and also by *syscalltracker*; and application-based logging with *syslog*.

4.1 OpenSSH analysis

We exemplify the use of the methodology with the OpenSSH attack and go through the steps in more detail for this attack. This attack is a local root attack against the OpenSSH server. The flaw is in the ssh RSA login procedure. The user adds an option to the RSA key, which makes the ssh server load a user-defined shared library. The flaw is that the shared library is loaded with the rights of the ssh server, and not the user. Figure 2 shows the main user actions in the different phases of the OpenSSH attack. For each of these user events, we created a meta-event by adding the expected system reaction and response.

4.1.1 OpenSSH events. The normal events we chose to compare the attack events to, were ssh password login for the RSA setup, RSA login session without commands for the key preparation and creation of shared library, and a normal RSA login with identity checking commands for the attack execution and system usage events. The failed attack variants used were an attack against an invulnerable OpenSSH version (3.1), and one against the vulnerable OpenSSH version (2.9) run with the "UseLogin" option set to "no" in the configuration file. The second variant does not use the systems /bin/login program, which is necessary for the attack to work. Table 4.1.1 shows an example of a meta-event, from the OpenSSH attack, with the corresponding normal and failed attack meta-event.

4.1.2 Comparing traces. The syslog files were uncomplicated to compare. There were few entries for each event and they always ended up in the same order.

The tcpdump logs were a little bit more complicated, since the packets did not always show up in the same order, and since they were encrypted it was difficult to match them. The comparison of tcpdump files were supported by

Table 1. A meta-event example for the OpenSSH attack

Events of one meta-event	Successful attack	Normal behaviour	Failed attack (wrong SSH version)
user action	SSH RSA login with prepared key	SSH RSA login with normal key	SSH RSA login with prepared key
system reaction1	"Installing" shared library	(no corresponding reaction)	(no corresponding reaction)
system reaction2	Using shared library	(no corresponding reaction)	(no corresponding reaction)
system reaction3	Performing identity check	Performing identity check	Performing identity check
system response	Root prompt	User prompt	User prompt

creating diff files created with the method described in Section 3.2 and by plotting packet sizes and TCP flags.

The system call logs were the most time-consuming to compare. Some system calls ended up in different order comparing two traces generated by the same event. These were brk(), rt_sigprocmask(), and wait4(). When the user were typing commands, the calls sometimes differed between runs since characters were sent one or two at a time. Still, it was feasible to use the diff files to extract the manifestations for all events in this attack.

4.1.3 OpenSSH attack manifestations and log elements. In the *syslog* logs there are no differences between successful attack and a normal ssh RSA login. Neither are there any significant differences in the tcpdump logs. The best tcpdump manifestation is the packet with the clear text ssh server version number. The traffic is encrypted, so the user commands etc can not be seen at all. The comparison of system call log files generate more useful manifestations.

Table 2 shows all the events for the OpenSSH attack and the presence of attack manifestations for the events in the different log files. Table 3 shows the chosen indicators for the OpenSSH attack, and the log elements needed for each of these indicators. Here it can be seen that we actually have strong manifestations from the syslog and tcpdump logs, but it should be noted that these are only from benign attack events, and thus not very useful for detection. From the system call logs we have strong manifestations of all events, except the actual "execution" of the shared library. We can only see the secondary effects of this event.

The first three indicators are from benign attack events, but are the best that could be created from syslog and tcpdump. The rest of the indicators are from malicious or indirectly malicious manifestations in the system call log. The fourth indicator uses the fact that the call setresuid() should not appear in the `login` program trace, i.e. it is a foreign symbol. The fifth indicator uses a sequence of system calls that do not normally appear. The sixth indicator uses the call getuid() with return value, and the rest of the indicators use a single system call together with a specific argument.

Table 2. Attack events for the OpenSSH attack

Attack events	Presence of attack manifestations: Syslog	Tcp	Syscall
Meta-event 1: [P1,N1,V2,M3]			
User action: Scp transfer of RSA key	weak	weak	strong
System reaction and response: -	-	-	-
Meta-Event 2: [P1,N1,V2,M3]			
User action: Password login to set up key	strong	strong	strong
System reaction and response: -	-	-	-
Meta-Event 3: [P1,N1,V2,M3]			
User action: Commands to set up key	no	-	strong
System reaction and response: -	-	-	-
Meta-Event 4: [P1,N2,V2,M3]			
User action: RSA login to test setup	strong	strong	strong
System reaction and response: -	-	-	-
Meta-Event 5: [P1,N1,V2,M3]			
User action: RSA login to do attack setup	strong	strong	strong
System reaction and response: -	-	-	-
Meta-Event 6: [P1,N1,V2,M2]			
User action: Commands to set up shared library	no	-	strong
System reaction and response: -	-	-	-
Meta-Event 7: [P1,N1,V2,M2]			
User action: Commands to prepare RSA key	-	-	strong
System reaction and response: -	-	-	-
Meta-Event 8: [P2,N1,V1,M1]			
User action: Ssh login using prepared RSA key	weak	weak	strong
System reaction 1: "Installation" of shared lib	no	no	strong
System reaction 2: Use of shared library	no	no	no
System reaction 3: Extra mprotect() calls	no	no	strong
System reaction 4: Lib debug info is printed	no	no	strong
System reaction 5: Changed setuid() calls	no	no	strong
System reaction 6: Extra egrep sessions	no	no	strong
System reaction 7: System checks identity	no	no	strong
System reaction 8: Root's PATH is used	no	no	strong
System reaction 9: Root's configurations files	no	no	strong
System response: User is given root prompt	no	no	strong
Meta-Event 9:[P3,N2,V1,M2]			
User action: "whoami" command	no	no	strong
System reaction 1: executes whoami program	no	no	strong
System reaction 2: checks user identity	no	no	strong
System response: answer to whoami	no	no	strong
Meta-Event 10: [P3,N2,V1,M2]			
User action: "id" command	no	no	strong
System reaction 1: executes id program	no	no	strong
System reaction 2: checks user identity	no	no	strong
System response: answer to id	no	no	strong

4.2 Tcpdump attack analysis

The Tcpdump attack generates interesting entries in both the network traffic log and the system call log. However, there are no entries from the attack in the syslog files. Also, there are no entries from the actual system violation. The stack operations are invisible in these logs; only the effects of the commands in the buffer overflow code can be seen. It is not obvious from the network traffic log that the started X11 session is caused by the attack, while it is striking in the syscall logs that the tcpdump program changes its execution behaviour drastically after receiving the attack packet. The processes involved in the attack is tcpdump, which executes "sh", and "xterm". Table 4 shows the events and presence of attack manifestations for the Tcpdump attack.

Meta-event 1 and 2 are both good events, even if event

Table 3. Summary of indicators and useful log data for OpenSSH attack

#	Indicator class	Event	Log	Log elements
1	[U4,C1]	E8(ua)	syslog	program name(sshd, login), time text(event, user, host)
2	[U5,C2]	E8(ua)	tcpdump header	src-port(ssh), TCP-flags, srcIP, dstIP, dst-port
3	[U4,C1]	E8(ua)	tcpdump w. data	src-port(ssh), data
4	[U2,C1]	E8(s5)	syscall name	process name(login), setresuid()
5	[U1,C1]	E8(s1)	syscall seq.	process name(login), open(), read(), fstat64(), old_mmap(), mprotect(), close()
6	[U2,C3]	E8(s7)	syscall w. ret	process name(bash), getuid() (or geteuid()) with return value + info to see that bash was started from login, with another user name: e.g. process name(login), pid(login, bash), execve(arg1,...), chdir(arg1)
7	[U1,C1]	E8(ua)	syscall w. arg	process name(sshd), read(..,arg2,..)
8	[U1,C2]	E8(ua)	syscall w. arg	process name(sshd), read(..,arg2,..), open(arg1,...)
9	[U1,C1]	E8(s1)	syscall w. arg	process name(login), open(arg1,...)
10	[U1,C1]	E8(s1)	syscall w. arg	process name(login), read(..,arg2,..)
11	[U1,C1]	E6(ua)	syscall w. arg	process name(bash), read(..,arg2,..)
12	[U1,C1]	E6(ua)	syscall w. arg	process name(bash), read(..,arg2,..)
13	[U1,C4]	E6(ua)	syscall w. arg	process name(sshd), write(..,arg2,..)
14	[U1,C1]	E7(ua)	syscall w. arg	process name(bash), read(..,arg2,..)
15	[U1,C1]	E7(ua)	syscall w. arg	process name(bash), read(..,arg2,..)
16	[U1,C4]	E7(ua)	syscall w. arg	process name(sshd), write(..,arg2,..)

Table 4. Attack events for the Tcpdump attack

Attack events	Presence of attack manifestations		
	Syslog	Tcp	Syscall
Meta-Event 1: [P1,N2,V1,M1]			
User action: Attack with wrong offset	no	strong	no
System reaction 1: receive packet	no	strong	strong
System reaction 2: decode packet	no	no	no
System reaction 3: system overwrites return addr.	no	no	no
System reaction 4: system uses new return addr.	no	no	no
System reaction 5: tcpdump crash	no	strong	strong
System response: no response	-	-	-
Meta-Event 2: [P2,N1,V1,M1]			
User action: Send buffer overflow AFS packet	no	strong	no
System reaction 1: receive packet	no	strong	strong
System reaction 2: decode packet	no	no	no
System reaction 3: system overwrites return addr.	no	no	no
System reaction 4: system uses new return addr.	no	no	no
System reaction 5: system executes attack code	no	no	strong
System reaction 6: Xterm with root shell started	no	strong	strong
System response: Xterm win. with root prompt	no	strong	strong
Meta-Event 3: [P3,N2,V2,M2]			
User action: "whoami" command	no	strong	strong
System reaction 1: executes whoami program	no	no	strong
System reaction 2: checks user identity	no	no	strong
System response: answer to whoami	no	strong	strong
Meta-Event 4: [P3,N2,V2,M2]			
User action: "id" command	no	strong	strong
System reaction 1: executes id program	no	no	strong
System reaction 2: checks user identity	no	no	strong
System response: answer to id	no	strong	strong

2 is slightly better since it is not necessary for the user to try the attack with the wrong offset before he succeeds. We have chosen to focus on the indicators for event 2 here. Table 5 shows the classified indicator and the log elements they require.

The first indicator is used to check packet length on incoming packets to port 7000. Probably, the length of this packet exceeds the normal length. The second indicator checks if the system answers the incoming packet to port 7000. The third indicator checks that an X session is opened. The fourth indicator is used to check if the content of packets to the 7000 port contains buffer overflow code. Number five is used to reveal that an xterm window is opened remotely, and number six reveals the root prompt the user gets. There are a great deal of system calls appearing in the sh and xterm processes that do not normally appear in the tcpdump log. These "unique" calls can be used one and one, or there are several sequences of other calls can be used.

Table 5. Summary of indicators and useful log data for the Tcpdump attack

#	Indicator class	Event	Log	Log elements
1	[U2,C1]	E2(ua)	tcpdump header	dst-port(7000), data-len
2	[U2,C2]	E2(s1)	tcpdump header	dst-port, src-port, srcIP, dstIP
3	[U2,C1]	E2(s6)	tcpdump header	dst-port(X), srcIP, dstIP
4	[U1,C1]	E2(ua)	tcpdump w. data	dst-port(7000), data
5	[U1,C2]	E2(s6)	tcpdump w. data	src-port(X), data
6	[U3,C1]	E2(sr)	tcpdump w. data	src-port(X), data
7	[U2,C1]	E2(s5)	syscall name	process name(tcpdump), execve()
8	[U2,C1]	E2(s5)	syscall name	process name(tcpdump), getuid() (or getgid(), geteuid(), getegid())
9	[U2,C1]	E2(s5)	syscall name	process name(tcpdump), setresuid() (or setregid())
10	[U2,C1]	E2(s5)	syscall name	process name(tcpdump), personality()
11	[U2,C1]	E2(s6)	syscall name	process name(tcpdump), access()
12	[U2,C2]	E2(s5)	syscall seq.	process name(tcpdump, sh,xterm), execve(), ...
13	[(U2+)U3, C1]	E2(s6)	syscall w. ret	process name(tcpdump,sh), pid, getuid() with return value
14	[U1,C1]	E2(ua)	syscall w. arg	process name(tcpdump), recvfrom(...,arg2,...)
15	[U1,C1/C2]	E2(s5)	syscall w. arg	process name(tcpdump,sh), pid, execve(arg1,...)
16	[U1,C1/C2]	E2(s6)	syscall w. arg	process name(tcpdump,sh), pid, execve(arg1,...)
17	[U1,C1/C2]	E2(sr)	syscall w. arg	process name(tcpdump,sh), pid,read(...,arg2,...)

4.3 Neptune analysis

The Neptune attack generates a great deal of interesting entries in the network traffic log, while both syslog and the syscall logs contain very few and not very useful entries from the attack. This may not be very strange since it is an network protocol attack. The attack does not activate any new processes and the only process that reveals any presence of the attack is the tcpdump process doing logging. The only thing that can be seen in syslog are a few entries from the preparatory events when the user connects to certain services. Table 6 shows the events and attack manifestations for the Neptune attack. Table 7 shows the classified indicators and the log elements they require.

The first indicator is used for studying incoming packet to the same port with the SYN and maybe ENC flags set, also source IP is useful to see if they come from the same

Table 6. Attack events for the Neptune attack

Attack events	Presence of attack manifestations		
	Syslog	Tcp	Syscall
Meta-Event 1: [P1,N2,V2,M3]			
User action: Connect to port to test if it is open	strong	strong	strong
System reaction: -	-	-	-
System response: Answer from service on port.	no	strong	strong
Meta-Event 1(v2): [P1,N2,V2,M2]			
User action: Portscan	weak	strong	weak
System reaction: -	-	-	-
System response: Answer from service on port.	weak	strong	strong
Meta-Event 2: [P2,N1,V1,M1]			
User action: Send 1000 attack packets	no	strong	no
System reaction 1: receive packets	no	no	weak
System reaction 2: conn. fill memory buffer	no	no	no
System response: no response	no	strong	no
Meta-Event 3: [P3,N2,V2,M2]			
User action: Test port	no	strong	no
System reaction 1: receive SYN packet	no	no	weak
System reaction 2: full buffer prevents answer	no	no	no
System response: no response	no	strong	no

Table 7. Summary of indicators and useful log data for Neptune attack

#	Indicator class	Event	Log	Log elements
1	[U1,C2]	E2(ua)	tcpdump header	dst-port, TCP-flags, srcIP
2	[U2,C2]	E2(sr)	tcpdump header	dst-port, src-port, TCP-flags, srcIP, dstIP
3	[U2,C2]	E2(ua)	syscall seq.	process name(tcpdump), recvfrom() or process name (all other), a selected collection of calls

computer. The second indicator can be used to check if a lot of SYN packets are coming in to the same port, and does not generate answers. The third indicator is used for checking if there are very many packets received by tcpdump, but few other processes are activated by these packets.

5 Discussion of attack analysis

The analysis of the tree attacks and the three types of log sources done in this paper shows that the methodology is feasible to use for different types of attacks and log sources. However, automation of the manifestation extraction would be useful, since this is rather time consuming.

5.1 Answering questions about the attacks

Some of the questions posed in Section 2 can be connected to specific events of the attacks. Also, there seems to be a relation between the questions and specific log elements. The questions *when did it happen?*, *who did it?* and *from where was it done?* can be answered in any attack phase and by any of the events. These questions can for example be connected to the syslog log elements date and time, host or IP, and user name. *Did someone attempt to do something malicious?* can be connected to the user action part of the meta-events in phase one and two (preparation and execution). *Did the malicious activity succeed?* is related to the system response and to some extent to the system reactions in attack phase two. *What effect did the attack have on the system?* is related to system response in attack phase two and three. The last question; *What did the attacker change in the system?* is related to the events in attack phase three.

In the system call logs, it seems that for example the calls *open()*, *read()* and *recvfrom()* has a connection to the question *did someone attempt to do something malicious?* The calls *setresuid()*, *getuid()* and *execve()* seems to often be involved in answering the questions of *did the malicious activity succeed?* and *what effect did the attack have on the system?*

In network traffic, source IP is the main tool for answering *from where was it done?*. The data part is necessary to find out more about *who did it?* Destination port and data part may be useful for answering *did someone attempt to do something malicious?* Not surprisingly, it is difficult to see the system reaction in network traffic, which means that it may be difficult to answer *what effect did the attack have on the system?*

5.2 Discussion of log sources

The usefulness of the information in the different log sources vary substantially. As we can see, *Syslog* is not very useful for any of the three attacks, *Tcpdump* and the *System call logs* are ineffective for one attack each.

Syslog is not really useful for detecting any of the studied attacks. Though, syslog can be of use for providing additional information through correlation, e.g. by providing user and host name for sessions to system call analysis.

A simple addition to syslog, would be information about the users effective user ID from the login process, which probably would make it useful for detection the OpenSSH attack. For the Tcpdump and Neptune attacks, we would need messages from the operating system kernel that are not easy to get.

Tcpdump is not possible to use for the OpenSSH attack, but it is very useful for the Neptune and Tcpdump attacks. For the Neptune attack, the packet header is the interesting part, while in the Tcpdump attack the data part is the most interesting. The log elements used in the three studied attacks are IP source and destination address, TTL, TCP source and destination port, TCP flags, data length, and the data part.

System call logs. For the Neptune attack, the system calls are not very informative, but for OpenSSH this is the only log revealing the attack, and it is also very useful for the Tcpdump attack. For both OpenSSH and Tcpdump it is possible to find single system calls that can be used for detection, even if the system call arguments reveal more of what is actually going on, and may be more useful for identifying the attack. The log elements of the system calls, process ID, process name, system call name, argument and return

value, are all useful in different indicators. The specific system calls that are part of the indicators for the studied attacks are execve(), open(), read(), write(), setresuid(), getuid(), geteuid(), chdir(), fstat64(), old_mmap(), mprotect(), recvfrom(), access(), and personality().

We have used both *Strace* and *Syscalltracker* in all our logging experiments. Both have some advantages and disadvantages, even though they mainly do the same job. *Strace* has the advantage of logging all system calls without exceptions, for a single process. It is also easy to follow the execution trace of a program and the processes it spawns, since no other processes interfere with it. Strace also shows the signals received by the process, *Syscalltracker* logs (almost) all system calls from all processes on the system. This is an advantage since it is not always obvious which processes that are affected by an attack. Syscalltracker logs provide both the process name and process ID and are very good for analysing which processes that are involved in an attack. A major disadvantage of syscalltracker is it's bad impact on the performance of the host machine.

Addition of user identity information for processes in the logs would probably improve the ability to detect some attacks. In our case, it would make it easier to detect the OpenSSH attack. The real and effective user ID may be added, and is already available in the Sun BSM system call logging tool [Sun00]. User commands are complicated to extract from the system call logs, because one letter is sent in each system calls with other calls in between. These would be easier to extract from a specialised user command log.

6 Conclusions

This paper presents a framework for determining the log data requirements for attack detection. We have suggested methods for extracting the log entries that differ between an attack event and the corresponding normal event and which can be used for attack detection. We have also shown examples of how indicators can be created from these attack manifestations, and how useful log elements can be defined from the indicators. The three attacks studied in this paper differ in their demands on log data. One is not possible to detect with *network traffic logs*, and one can not be detected with *system call logs*. None of them can be detected with only the information from *syslog*. This work presents a good starting point in the search of a new information source for attack detection and may be of help in the construction of intrusion detection signatures. Automation of the attack analysis is part of our ongoing work. Also, further attacks should be studied to find classes of attacks with different demands on log data and further log sources should be studied to find the information that is most convenient to log at the same time as it reveals as much as possible of the attack.

7 Acknowledgements

Thanks to Magnus Almgren and Ulf Larsson for valuable discussions and suggestions in the construction of this paper.

References

[ADD00] M. Almgren, H. Debar, and M. Dacier. Lightweight tool for detecting web server attacks. In *Proceedings of the Network and Distributed System Security Symposium*, 2000.

[ALGJ98] Stefan Axelsson, Ulf Lindqvist, Ulf Gustafson, and Erland Jonsson. An approach to UNIX security logging. In *Proceedings of the 21st National Information Systems Security Conference*, Arlington, Virginia, USA, 1998.

[ATS+03] Christina Abad, Jed Taylor, Cigdem Sengul, William Yurcik, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrusion detection: A proof of concept. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, NV, USA, 2003.

[HA93] Stephen E. Hansen and E. Todd Atkins. Automated system monitoring and notification with Swatch. In *Proceedings of the Seventh Systems Administration Conference (LISA '93)*, Monterey, CA, 1993.

[KMT04] Kevin S. Killourhy, Roy A. Maxion, and Kymie M. C. Tan. A defence-centric taxonomy based on attack manifestations. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2004)*, Florence, Italy, June 2004.

[Kum95] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, West Lafayette, IN, USA, August 1995.

[LP01] Ulf Lindqvist and Phillip A Porras. eXpert-BSM: A host-based intrusion detection solution for Sun Solaris. In *Proceedings of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, 2001.

[Max03] Roy A Maxion. Masquerade detection using enriched command lines. In *International Conference on Dependable Systems & Networks (DSN-03)*, San Francisco, California, 2003.

[Roe99] Martin Roesch. SNORT - lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference - LISA '99*, Seattle, Washington, USA, 1999. USENIX.

[Sun00] Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA, USA. *SunSHIELD Basic Security Module Guide*, February 2000. `http://docs.sun.com/db/doc/806-1789`.

[VRKK03] G. Vigna, W. Robertson, V. Kher, and R.A. Kemmerer. A stateful intrusion detection system for world-wide web servers. In *Proceedings of the Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, 2003.