;login:

THE MAGAZINE OF USENIX & SAGE

November 2000 • volume 25 • number 7

{

THEME ISSUE: SECURITY

edited by Rik Farrow

inside:

CORRELATING LOG FILE ENTRIES

#

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

correlating log file entries

by Steve Romig

Steve Romig is in charge of the Ohio State University Incident Response Team, which provides incident response assistance, training, consulting, and security auditing.

<romig@net.ohio-state.edu>

I have often needed to peruse log files from different systems while investigating computer crime, performance issues, and other odd happenings – and I've learned a few tricks that I'd like to share with you. The general principles will apply to most investigations, but I'll draw my examples mostly from the UNIX and incident-response worlds with which I am most familiar.

I've written most of this while sitting at Camp Ohio, a 4-H camp, where I'm volunteering as a counselor for my church's junior high summer camp. Trying to write an article on such a technical subject between archery and setting up a campfire and night time zip line is an interesting challenge (a zip line is where you jump off a tower suspended below a cable by pulley and harness and ride the cable down to the ground some distance away – imagine doing that in the dark!) Between my co-counselor Marco and myself we had more computing power with us than the rest of the camp combined, but amazingly our cabin still didn't win the "geekiest cabin" award the day that was the theme for cabin clean-up. Maybe if we had had a working Internet connection . . .

Let's suppose that you are investigating a compromised computer, and you are fortunate enough to have tracked the activity back to the source and have access to all of the systems involved. In our case, a suspect used his home computer to connect to the Internet through our modem pool using a stolen account. Once on the Internet, he used a variety of tools to probe for and break into victim hosts for various purposes. (See Figure 1.)

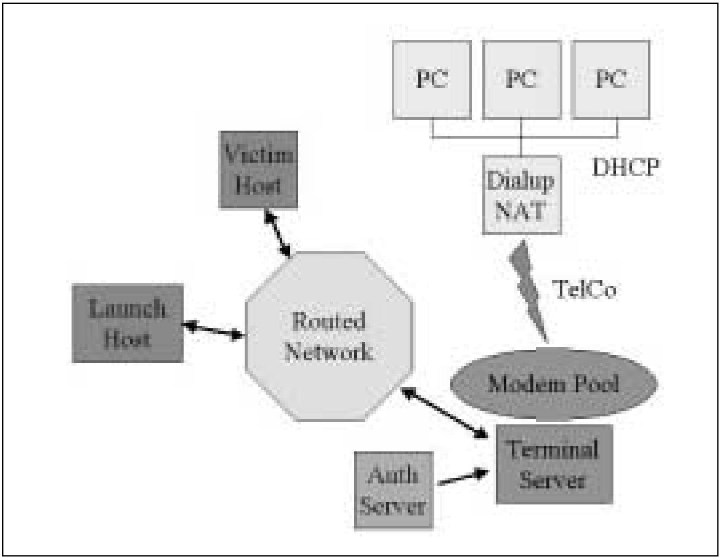

Figure 1

One common goal in these sorts of investigations is to reconstruct a chronological record of events and a list of other facts. Once we have done that, we develop one or more theories that account for this history and set of facts. If we are working on the side of the prosecution in a computer-crime investigation, our prime theory would be something along the lines of "the butler did it with mstream in the kitchen." If we are working on the side of the defense, our theory might be "the prosecution's theory didn't account for this and that evidence that shows that the butler couldn't have done it." The supporting evidence and these theories are presented before the court and the jury ("the trier of fact") is called upon to determine whether the prosecution has sufficiently proven its case or not. Obviously, how well we can construct the record of events and fit the pieces together has great bearing on the outcome of the investigation.

We need to consider several issues. First, we need to be proficient at finding the evidence. If you can't find the evidence in the first place, you'll have a hard time fitting it into your reconstructed chain of events. We also need to understand what the evidence actually means. If we misunderstand the evidence, then either our reconstruction will be wrong or we'll create faulty theories that explain the evidence. Finally, we need to understand how to piece evidence from different sources together to create a cohesive reconstruction. If we know where the evidence can be found, what it means, and how it fits together, then we'll be well on our way to reconstructing the chain of events. Note that I am totally ignoring issues concerning preservation of evidence for use in a civil or criminal trial. Sorry!

Know Where the Evidence Is

I won't dwell on this here – full treatment of the subject is way beyond the scope of this short article. In general, this means that you have to know where evidence pertaining to your case *might* be, and then look to see whether you can actually find it. For instance, in our example investigation, we might find evidence in the following locations (look back at Figure 1):

Home system	Dial scripts, dial logs, files containing output from exploit tools, lists of compromised hosts, etc.
Phone system	Phone traces or pen registers
Modem pool	TACACS, TACACS+, or RADIUS authentication logs
Networks	logs of network activity, such as Cisco Netflow logs or from the use of tools like Argus
Victim and intermediate hosts	Syslog records showing access to network services through TCP wrappers or other means; login records such as utmp, wtmp, wtmpx (or in syslog if you are smart enough to use loginlog, a program that transcribes wtmp entries to syslog); processes running on the system (and the associated memory, binaries, network connections, and files); free and slack space on the filesystem, and so on.

Think about the components involved in the incidents you are investigating – what information might they contain? If you don't know enough about them, it doesn't hurt to find an expert and ask questions. Many people fail in their investigations because they fail to ask questions about the components involved and thereby miss important evidence.

What the Evidence Means

It is relatively easy to understand where the evidence might lie. Draw a block diagram of the system under investigation and consider each component in turn – that at least gets you a high level view. Understanding what the evidence actually means is trickier. For one thing, it involves a deeper understanding of the component systems involved. At the very least, we need to understand how the evidence is created or compiled – for instance, knowing that the UNIX login program (and some others, like sshd) updates the wtmp/wtmpx/utmp logs and under what circumstances.

Knowing what the evidence means helps us avoid conclusions that aren't logically supported by the evidence. For example (and pardon me if this seems simplistic), a TACACS log entry that indicates that the "romig" account logged in means just that – the "romig" account was used to log in. It does not prove that the owner of the account was the one who used the account to log in, although the theory that "Steve Romig, the owner of the romig account, used it to log in at this time" is consistent with this evidence. Similarly, a DHCP (Dynamic Host Configuration Protocol) server log that shows that a host with a particular MAC address had a lease for a given IP address does not mean that that host was the only host using that IP address during that time period; it just means that this host held the lease. The theory that "this host held the lease for this IP address at the time and used that address to probe the victim" is consistent with the lease evidence, but the lease evidence doesn't conclusively prove this theory, since there are other plausible theories that are also consistent with this evidence.

Understanding what the evidence means also helps us recognize potential blind spots. One modem pool that I worked with used a pair of authentication servers handling authentication requests in a round-robin fashion. This meant that log entries pertaining to login/logout events for any given terminal server port could be found in the logs from either server. If we only looked at the records from authentication server A (see Figure 2), we might mistakenly conclude that the "romig" account was used to authenticate the session that spans 1:15:21 (the time that some nefarious Internet crime

time	authentication server A	authentication server B
1:02:12	login - romig	
1:10:32		logout
1:10:56		login - farrow
1:26:09	logout	

Figure 2: Login/logout events for a single port on a terminal server.

occurred, which we traced back to this terminal server port). Note that in this example the logout records do not name the associated account name that goes with the corresponding login records. You need to merge and sort the logs from both servers before you can reconstruct an accurate history of login/logout events.

Again, don't be afraid to get help from an expert.

How It Fits Together

When we conduct an investigation we collect bits and pieces of information from various sources. These sources vary in completeness and in reliability. The real point to this article is to talk about how to correlate the pieces together. When we do this we commonly run into several problems.

TIME-RELATED ISSUES

First, let's talk about the time-related issues. Most log files include some sort of timestamp with each record, which can be used to correlate entries from several logs against one another. One common problem we run into when correlating logs from different hosts together is that the clocks on those hosts may not be synchronized to the same time, let alone the correct time. You can sometimes infer this clock offset from the logs themselves. If the shell history file for my account on host A shows me running "telnet B" at time T1, but the TCP wrapper log on host B shows the Telnet connection at T2, then we can conclude that the clock offset between host A and host B is roughly T2-T1 (assuming they are in the same time zone). It isn't always possible to infer this offset directly, since there can be a significant lag between events in different logs (see below).

It is also important to know the time zone that each log was recorded in. Unfortunately, the timestamps in many logs do not include the time zone. Get into the habit of sending time-zone and clock-correction information when you send logs to others, and request the same when you ask others to send logs to you. I generally like to express time zones as offsets from GMT, since that is more universally understood and is less ambiguous than some of the common abbreviations.

Event lag is the difference in times between related events in different types of logs. For example, suppose that someone connects from host A to host B using Telnet and logs in. A Cisco Netflow log containing the traffic between A and B will record the time T that traffic to port TCP/23 (typically Telnet) on host B was first seen. If host B uses TCP wrappers to log access to the Telnet service, the log entries for that entry will probably have a timestamp very close to T. However, there can be a considerable delay between when a person is presented with a login prompt and when she actually completes the authentication process, which is when the `wtmp` record would be created. So I might see a NetFlow entry indicating attempts to connect to the Telnet service at 13:02:05, a TCP wrapper entry at 13:02:05, and a login entry at 13:02:38, 33 seconds later.

Event lag is important because often our only means of correlating entries from different logs together is through their timestamps. Unfortunately, since the amount of lag is often variable, we can't always correlate events specifically by starting time or even duration since the session in the network-traffic log would last longer than the login session. However, we can use session duration and starting time to eliminate false correlations – a login session that lasts 0:23:32 wouldn't (usually) match a phone session that lasts only 0:05:10. We can sometimes use the ending time of a session to make closer correlations, since the ending events often match up more closely in time. For example, logging out of a host you connected with telnet usually ends the Telnet session and its associated network traffic, so the logout event and the end of network traffic in the NetFlow log would be very close chronologically.

Sometimes logs are created in order of the ending time of a session, instead of the start time. This can lend further confusion to the correlation process. Log entries for Cisco Netflow logs are created when the "flow" of traffic ends. UNIX process accounting logs are created when the associated process ends. It is easy to misinterpret such logs, since important information may be buried much later in the log. Figure 3 shows the process accounting records corresponding to a login shell where someone ran ls, cat, and then a shell script that ran egrep and awk. Note that the sh processes corresponding to the login session and the shell script that was run show up after the processes started from within those shells. If you were just casually reading the log, however, you might miss this – I know I have on several occasions, and was very confused until I realized my mistake. Note that not all systems provide tools that print process accounting records in this format – the basic data is there in the file, but you might have to write some software to winkle it out!

```
                    start
line   account      time        duration     command
———    ————-        ———         —————        ————-
ttyp1 romig         12:32:28    00:00:07       ls
ttyp1 romig         12:33:02    00:00:05       cat
ttyp1 romig         12:33:45    00:00:03       egrep
ttyp1 romig         12:33:45    00:00:04       awk
ttyp1 romig         12:33:45    00:00:04       sh
…
ttyp1 romig         12:20:12    00:10:02       sh
```

Figure 3: Process accounting records.

We can often can use the time bounds on one session to "focus in" on smaller portions of other logs. For example, if the modem-pool authentication records show a login session starting at 07:12:23 and lasting for 00:12:07, we can narrow our search through things like process accounting logs and other logs on target systems to just that time range (assuming that we've corrected for clock offsets and time zone). That's fairly straightforward, and we do this sort of bounding naturally. What may not be obvious is that we cannot always do this. Most of the log entries associated with a login session on a host should fall within the start and end times of that session. However, it is easy to leave a process running in the background so that it will persist after logout (using nohup), in which case its process accounting records will not be bounded by the login session.

MERGING LOGS

We sometimes have to merge logs made on different systems together to build a complete picture. For instance, on some occasions we have set up authentication servers that operate in parallel, in which case logout records may not be left on the same server that handled the corresponding login record. The Ohio State University now has two different routers that handle traffic to different parts of the Internet. There are some hosts where network traffic goes out through one router and returns through the second (due to asymmetric routing). If we are looking through Cisco Netflow logs for traffic, we now need to be careful to merge the logs together so that we have a more complete record of network activity. This can also be an issue in cases where we have multiple SMTP servers (records of some email will be here, some there) and for Web proxy servers.

RELIABILITY

Logs vary in the degree to which they can be relied upon to be accurate recordings of "what happened." Their reliability hinges on issues like the ownership and mode of the log files themselves. For instance, the utmp and wtmp logs on some UNIX systems are world-writable, meaning that anyone on the system could modify their contents. We are also dependent on the integrity of the system pieces that generate the logs. If those subsystems have been compromised or replaced, the logs that they generate may not be

Sometimes it isn't what we find in the log that is interesting, but what we don't find.

a complete or accurate portrayal. If an intruder has replaced the login binary with a "rootkit" version that doesn't record login entries for certain users, then the login logs will naturally be incomplete.

In other cases, the accuracy of the logs is subject to the security of the network protocols used for transporting the messages. Syslog and Cisco Netflow logs are both sent using UDP (the User Datagram Protocol), which makes no provisions to ensure that all data sent will be received. In these cases the logs can easily be incomplete, in the sense that records that were sent from the source were never received by the server that made the record that we are examining. This also means that it is relatively easy to create false log entries by directing carefully crafted UDP packets with spoofed source addresses to the log servers.

We can help guard against the dangers of incomplete or incorrect logs by correlating events from as many sources as possible. We will still have to adjust our theories to account for discrepancies among the logs, but at least these discrepancies will be more visible. This is especially true in the cases where system processes on a host have been modified or replaced by an intruder.

IP ADDRESS AND HOST NAME PROBLEMS

We need to realize that IP addresses can be spoofed and recognize cases where this is likely and cases where it is unlikely. (For example, spoofing is common in flooding attacks and rare for straight Telnet connections.) There is also a variety of games that people can play to steal domains, poison the caches on DNS servers, and otherwise inject false information into address/name lookups.

Unfortunately, many subsystems resolve the IP addresses that they "know" into names using DNS, and then only log the resolved names, which may not be correct. So we also need to recognize that the host names that we see in log files may not represent the correct source of the traffic that generated the log message. It's generally best for log messages to include both the IP address and the name that it was resolved to, rather than one or the other. If I had to choose one, I would choose the IP address, since that's more correct in most contexts (in the sense that the subsystem "knows" that it saw traffic with a source IP address of A.B.C.D, and we can't know whether the resolved host name for that is correct).

RECOGNIZE WHAT'S MISSING

Sometimes it isn't what we find in the log that is interesting, but what we don't find. If we see NetFlow data showing a long-lasting Telnet session to a host but no corresponding login entry for that time period, this should naturally raise the suspicion that the login entries are incomplete (or that the NetFlow data was incorrect). If a shell history file shows that someone unpacked a tar archive in /dev/ – but we cannot find /dev/ on the system – then someone has either deleted it or it is being hidden by a rootkit of some sort.

Some Comments on Specific Logs

I have a few parting comments about some of the logs that we commonly work with, in light of the issues that I've addressed in this article.

PHONE LOGS

I don't know whether the phone companies do anything to synchronize the clocks used for timestamping phone trace logs; past experience shows that they are usually close to correct, but are usually off by a minute or two. Note also that there can be significant event lag between the start of a phone connection and the start of an authenticated session on the modem pool that someone is connecting to (or start of activity in other logs). The easiest way to match calls to login sessions and other logs is by narrowing down the search by very rough time constraints and especially by call duration. We tend to have many short dialup sessions and relatively few long sessions, and so it is generally easier for us to match login sessions against longer phone calls, since they are "more unique" than the shorter calls. For example, there are few calls that last at least 2:31:07, but many that last at least 00:05:21.

Note that the most we can say in these cases is that the process-accounting records are consistent with running the script "blub."

UTMP, UTMPX, WTMP, AND WTMPX LOGS

Apart from the reliability concerns mentioned above, on some UNIX systems you also run into problems that are due to the fact that the wtmp and utmp files truncate the source host name (for remote login sessions) to some limited size. This obscures the source host name if it is long. One way to help address this is to use other sources (like TCP wrapper or network traffic logs) to try to determine the correct host name.

UNIX PROCESS-ACCOUNTING RECORDS

One problem with process accounting records is that they only contain the (possibly truncated) name of the binary that was executed, and not the full pathname to the file. Consequently, to find the binary that belongs to a process accounting record, we need to search all attached filesystems for executable files with the same name. If there is more than one file, it may not be possible to specifically determine which binary was executed. In the case of shell scripts, the name of the interpreter for the script is recorded (e.g., Perl, sh, ksh), but the name of the script isn't recorded at all.

In some cases we can infer the name of the executable on the basis of other records, such as shell history files and by examining the user's PATH environment-variable settings. If we see from a user's shell history file that a command named "blub" was run at a given time, and a search of attached filesystems reveals a shell script named "blub" in a directory that lies in their "PATH," we can reasonably correlate the file with the shell history file entry and the process accounting record for the shell that was invoked to interpret the contents of "blub." We should be able to make further correlations between the contents of the script "blub" and the process accounting record if the script executes other programs on the system. This is especially true if the sequence of commands executed is unique, or the commands are not commonly used in other places. Note that the most we can say in these cases is that the process accounting records are consistent with running the script "blub." We cannot prove directly from the process accounting records that the script was what generated those log entries – for instance, a different script named "blub" might have been run, and then deleted or renamed.

UNIX SHELL HISTORY FILES

Some UNIX shell history files are timestamped – otherwise, it can be very difficult to match these records to other events, such as process accounting records. Note, of course, that shell history files are typically owned by the account whose activity they record, and so are subject to editing and erasure. You should be able to match the events depicted in the shell history file against the process accounting records and sometimes against others, like logs of network traffic, timestamps on files in the local filesystem, and so on. The shell history is written when each shell exits, so overlapping shells can obfuscate the record. (History is written by the last to exit. . . .)

There's a wealth of information available in other logs on a system, especially if the log levels have been tweaked up by a knowledgeable administrator.

SYSLOG, NT EVENT LOGS, AND OTHER TIMESTAMPED LOGS

There's a wealth of information available in other logs on a system, especially if the log levels have been tweaked up by a knowledgeable administrator. Take note of my cautions above about correlating log entries by timestamps and about the reliability of the logs. It is ideal if you can log to a secure logging host so that an intruder can't easily modify previously logged events. This is easy to do with syslog, and fairly easy to do with NT event logs using both commercial and free software. There's even software that allows you to "transcribe" NT event-log entries to a syslog server. One thing to beware of – with syslog, the timestamp that appears on the entries in the log file is the time that the entry was received by the local machine according to its own clock, not the clock of the machine that the log entries come from. That's generally a good thing, since you've hopefully taken pains to synchronize your syslog host's clock to "real time." However, it can cause confusion if you try to correlate those log entries to other events from the original host, since there may be a clock offset between that host and the syslog host.

OTHER SOURCES THAT WE HAVEN'T TALKED ABOUT

There's a wealth of information that can potentially be found on the local host – binaries, source code, output from commands run, temporary files, tar archives, contents of memory of various processes, access and modification times for files and directories, files recovered from the free and slack space on the filesystems, information about active processes, network connections and remote filesystem mounts at the time of the incident, etc. You need to hunt for these and fit them into your reconstruction of the history of the event. For most of this information, unless you have access to more detailed logs (e.g., timestamped shell history files or tcpdump captures of the Telnet session where the intruder did his work), a lot of this reconstruction will necessarily be informed guesswork. Suppose we find a process running on a UNIX host and run lsof on it. (lsof lists the file handles that a process has open – very handy for investigations where processes have been left running.) If lsof reveals that this process has open network connections, we might be able to correlate these against entries from network traffic logs based on the time, the host's IP address, the remote IP address, the IP protocol type, and the UDP or TCP port numbers (if applicable).

Take-Home Lessons

There are a few practices you can follow to improve the condition of your logs and make it easier to correlate them against one another. First, turn your logging on and log a reasonable amount of data (both in quality and in quantity). Disks are cheap these days, so you can afford to both log more and retain it longer. It is always a good idea to forward copies of your logs to a secure log server – this is easy to do with both syslog and NT event logs. Synchronize your clocks to a common source – if you don't want to synchronize them to an external source, you can at least set up a fake internal source and synchronize them using the network time protocol. If you have a choice, log IP addresses in addition to (or instead of) the host name that corresponds to the address – the host name might be more meaningful to you, but the IP address is more correct. Finally, secure your systems so that you don't have to do these sorts of investigations often!