

Intrusion Detection Data: Collection and Analysis

Robert F. Erbacher and Bill Augustine
Department of Computer Science, LI 67A
University at Albany-SUNY
Albany, NY 12222, USA

E-Mail: erbacher@cs.albany.edu and syswaa@cs.albany.edu

Abstract

Identifying intrusions, misuses, and attacks in general, require that systems be monitored for anomalous behavior. This includes the online analysis of system log files, system activity statistics, and user connection activity. Many intrusion detection and firewall tools will add to the volume of information that must be analyzed. The first step in any such tool is the collection and correlation of the relevant information and events from the system. This aspect of the activity has not been described and is performed to different extents by any particular tool. We describe the relevant information that can be found on most UNIX based systems and how to collect this information using Bourne scripts. We also describe why the collected information is relevant to the analysis task at hand and how this data can be correlated and analyzed.

Keywords: Computer Security, Intrusion Detection, Data Collection

1. Introduction

With current concerns over computer security, particularly related to network intrusions, misuse, and even cyber-terrorism, there is a need for tools to monitor and analyze systems to identify when an attack is occurring. This raises the issue of identifying exactly what information is available on a system for aiding in such an analysis. Given that the data *is* available, what does it mean, and how can multiple events from the data be correlated for a greater analysis capability? It is important that we focus on only typically available data. Work is in progress to modify systems at the kernel level for the

collection of security related data or to internally identify intrusions. However, these systems are generally not deployed and the capabilities to protect and monitor systems is needed immediately. Many system configuration parameters can be modified unobtrusively to collect additional information that can aid in the intrusion detection process without requiring a modified kernel. Other tools and packages can also be installed to collect additional information to aid in the analysis process.

This paper discusses our experiences with the collection of system related information. While we focus on information on UNIX based systems, as these systems are the main focus of actual intrusions, such attacks are becoming a concern on MS Windows based systems, which tend to be targeted for other types of attacks. When attempting to identify what information should be collected it is helpful to understand what the data means and more importantly what it means in correlation that other collected data, which can greatly enhance the meaning of each event. Furthermore, we discuss strategies for collecting such data which in many cases is not as straight forward as would be expected. In conjunction with determining what data should be collected, an issue of equal relevance is determining what data should *not* be collected as system logs provide an enormous amount of data and not all of it will be useful for the intrusion detection task.

2. Previous Work

Many tools are being made available to assist in the monitoring of systems for potential attacks and misuse. This includes such tools as BlackIce [1], PortSentry [2], TripWire [3], Cops [4], etc. However, these tools generate additional information which needs to be analyzed in conjunction with the typical system logs. There are insufficient capabilities available for correlating the enormous amounts of information being made available. We are working to develop visualization techniques [5] to aid in the processing of system log files, including events added to the system log files by other tools.

There has been no discussion as to how to effectively collect relevant information from a system, parse relevant events and parameters out of log files for further analysis, or correlating the information from multiple systems. Providing an understanding of exactly what information is available, how it can easily be collected, and why it is relevant is critical for the wider enhancement of security and intrusion detection related tools.

3. Data Collection

This work derives from our original efforts to collect intrusion related data using the Hummer Intrusion Detection System [6]. This environment is geared towards the collection of data from remote systems and the storing of said data on a specified server. The current version of the environment is written in java which resulted in several issues that needed resolving:

1. System administrators had difficult installing the environment. Lack of documentation and support led to difficulty in determining exactly how the environment should be installed and linked to a database correctly.
2. Lack of availability of Java. Many systems that we wish to collect data from do not have java installed. This is principally a problem with older systems and servers.
3. Lack of support for varying storage schemes and mechanics. Many systems store their log files in varying locations and formats that makes it difficult to use Hummer. We are also interested in exploring storage mechanisms other than postgres, such as LDAP [7].

4. Inability to enhance. Lack of access to original sources means that we cannot add additional data collection mechanisms, monitor or modify performance characteristics, explore encryption and compression protocols, etc.
5. Unknown performance characteristics. With the volume of data being collected the impact of the environment on the monitored system and the network interconnectivity is crucial. We must examine the impact on these characteristics and identify solutions for unacceptable performance characteristics.

3.1. System Log Files

As indicated earlier, much of the collected data is garnered from system log files. System log files consist of most of the pertinent information needed for intrusion detection. However, the information provided can be further enhanced with the appropriate configuration. When identifying event types we normally refer to the overlying identifier type, usually appearing at the very beginning of a message. For each event type we show a single example of what the event may look like in a normal system log. In reality, the actual events will take on a much greater variety of formats representative of the variety of different events that can engender the corresponding logged event. The system log files will generally provide the following principal event types that are of interest:

Initial port connection requests: On many systems, notably SUN systems, this requires that `inetd` [8] be run with the `-t` option. This provides details as to when an initial connection request was made, before the authentication process. This can be used to identify failed connection attempts. Systems with `xinetd` [9], an enhanced `inetd` utility, can generate more specific data indicating when a connection has failed to authenticate. It should be noted that failing to authenticate differs from not authenticating as a user may be connecting to the port to see if the service is available for later reference. Such messages will have an identifier within the message such as `inetd[#####]:` or `xinetd[#####]:`, where the `#####` refers to the connection identifier.

This option provides details that are essentially provided by tools such as `portsentry` [2] but

for active ports. These initial port requests when correlated with authentication success and user connection activity can aid in identifying expected vs. anomalous behavior. Additional parameters in the message identify the port or type of connection initiated.

```
Feb 17 04:10:45 acunix1.albany.edu
inetd[170]: [ID 317013 daemon.notice]
shell[5832] from 999.999.9.999 525
```

Syslog messages: Such messages occur when the syslog daemon restarts or cycles the log files. This is critical should an attacker be attempting to clear their tracks, though carelessly.

```
Feb 19 17:00:31 Visualizer-S syslogd
1.4-0: restart.
```

Portmap messages: Identifies calls to services by remote hosts, through the RPC mechanism, that are either not active on the given system or that are being accessed by a host that does not have permission to be accessing said service. The example below shows the effect of a broadcast message typical of many UNIX configurations. However, it is also quite common to see actual attacks which attempt to initiate a service believed to have a security hole.

```
Feb 19 10:31:42 Visualizer-S
ortmap[19413]: connect from
999.999.9.99 to callit(ypserv):
request from unauthorized host
```

Portsentry identified port scan logs: Portsentry is essentially a firewall based product that identifies attempts to connect to ports that do not have open services. These port probes are used by attackers to identify what type of system is being looked at and what services the system has available. This goes a long way towards providing an attacker with the information needed to apply a known attack or security loophole.

```
Feb 20 22:48:25 Visualizer-S portsen-
try[1432]: attackalert: SYN/Normal
scan from host:
address1.address2.type/999.99.99.999
to TCP port: 80
```

Sudo super user access: The sudo command allows users to perform super user or root based administrative tasks without actually becoming the root user. When used, sudo requests that the user identify themselves with their password. Should they not use the correct password or not

be in the sudo database then the attempt is logged as an indication of an individual attempting to subvert the root privileges of the machine.

```
Feb 20 22:53:02 Visualizer-S
sudo(pam_unix)[3937]: authentication
failure; logname=coolid uid=0 euid=0
tty=pts/0 ruser= rhost= user=coolid
```

Login messages: Login messages indicate when users have failed to successfully authenticate with their correct password. This can occur when a user forgets their password, a common event in an academic environment. It can also be the indication of someone attempting to crack a password using a brute force strategy, also a common activity in an academic setting. A small number of such events generally would not raise an alarm but if they occurred at high frequency it could be an indication of a brute force attack.

```
Feb 17 21:00:10 acunix1.albany.edu
login: [ID 760094 auth.crit] REPEATED
LOGIN FAILURES ON /dev/pts/14 FROM
alb-99-99-999-999.nycap.rr.com
```

Kernel messages: Kernel messages identify activity taking place at the kernel level. This will include typical noise but also identify when NFS connections are lost and gained, when packets are blocked through TCP Wrappers, shutdown and startup messages, and daemon startup messages. An enormous number of kernel messages can be generated on some systems but they can identify obtrusive systems, problematic network connections, and failing NFS servers.

```
Feb 20 20:54:53 Visualizer-S kernel:
Packet log: input DENY eth2 PROTO=6
99.999.99.99:2314 169.226.2.230:80
L=48 S=0x00 I=22575 F=0x4000 T=114
SYN (#1)
```

Switching users: The su command allows users to switch from one login identifier to another. Very rarely would such activity be expected by normal users of a system and frequent activity or attempted activity can be indicative of anomalous behavior, particularly if the attempt fails. In conjunction with identifying standard users switching logins this command can also be used to switch to the super user or root login identifier. Frequent such attempts are generally noted harshly. Generally, only failed attempts are noted in the log files.

```
Feb 20 22:39:33 acunix1.albany.edu
su: [ID 810491 auth.crit] 'su newid'
failed for falsid on /dev/pts/62
```

Daemon Messages: Daemon messages include all messages returned by the various daemons responsible for handling connections through the different connectivity protocols, including: telnetd, ftpd, and sshd. These daemons will place messages into the system log when a user has failed to successfully authenticate, when the authentication succeeds and the user logs in, when the user logs off or discontinues the session, and when an unexpected error is encountered through a session. Which messages are actually incorporated into the system log is daemon and configuration dependent. Notably, most daemons will not place connection and disconnection information into the message log as it is duplicated in the last log. Some daemons such as sshd will place connection and disconnection information into the *system* log. The authentication failures are of particular interest.

```
Feb 18 16:26:18 acunix1.albany.edu
in.ftpd[4]: [ID 120637
daemon.warning] falseid LOGIN FAILED
[from alb-99-99-999-999.nycap.rr.com]
```

The Bourne script commands for checking the message types described in this section are shown in figure 1. It consists of an initial check to

identify where the message log file is being stored. We have found the file in one of two principal directories, /var/log/messages and /var/log/syslog. The script then retrieves the end of the logfile that was added since the tool was last instantiated, parses out lines containing the appropriate keywords, converts the symbolic date and time information into a format suitable for the postgres database, and executes an SQL command to send the appropriate table entry to the database server.

The system will provide additional information that is not of particular relevance, namely e-mail transmittals and printer activity. Though many systems provide the ability to store the e-mail transmittal messages in a separate log file, others do not provide this capability or it is not used by the system administrators. This adds clutter to the system log and requires additional filtering when the log must be read manually.

3.2. On-line Statistics and Events

Since we are collecting data in an online fashion, additional information is available. In particular, we can collect statistics as to the usage of the system and the behavior of the network connectivity. While the examples described in the previous section merely report events occurring on the system, the facilities described in this section are not event based but are rather

```
msglogfile=[ -f /var/adm/messages ] && msglogfile=/var/adm/messages
[ -f /var/log/messages ] && msglogfile=/var/log/messages

function SysTool {
tail +"$LAST_SYSCOUNT"1 $msglogfile >/usr/tmp/tmpc.`hostname`
LAST_SYSCOUNT=`wc -l $msglogfile | tr -s [:space:]|cut -d\ -f 2`
string=$( $awk_prog
'/(su:|kernel|login|sudo|xinetd|portsentry|portmap|inetd|syslogd|telnetd|ftpd|
sshd|rshd)/ { gsub("\'\'\'", "\\\'\'\'"); printf("\'%s,%s,%s,%s\' ", $1, $2, $3,
substr($0,index($0,$5),length($0))};' /usr/tmp/tmpc.`hostname` )
eval "list=($string)"
  for message in "${list[@]}"
  do
    cormon=`echo $message|cut -d\, -f 1`
    corday=`echo $message|cut -d\, -f 2`
    cortime=`echo $message|cut -d\, -f 3`
    corcomm=`echo $message|cut -d\, -f 4`
    monno=$(echo -e "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug\nSep\nOct\nNov\n
Dec\n" | sed -n '/'$cormon'/=' )
    cordate="$monno-$corday-$(date +%Y)"
    eval $PSQL -c "INSERT INTO log VALUES\($2,3,4,\'$cordate\',
\'$cortime\',\'$SYS:$corcomm\',\'$ipaddr\',\'$msglogfile\', \'System\'\\)\;"
  done
}
```

Figure 1: Script code for parsing and collating daemon messages.

```

dfcomm=0
df -kP >&/dev/null
[ $? -eq 0 ] && dfcomm=1
df -k >&/dev/null
[ $? -eq 1 ] && dfcomm=2

function BDFTool {
[ $dfcomm -eq 0 ] && string=$(df -k | awk 'NR != 1{printf("\%s %s\n",
,$1,substr($5,1,length($5)-1));}')
[ $dfcomm -eq 1 ] && string=$(df -Pk |awk 'NR != 1{printf("\%s %s\n",
,$1,substr($5,1,length($5)-1));}')
[ $dfcomm -eq 2 ] && string=$(df | awk 'NR != 1{printf("\%s %s\n " ,
,$1,substr($5,1,length($5)-1));}')
eval "list=($string)"
  for filesys in "${list[@]}"
  do
    eval $PSQL -c "\"INSERT INTO log VALUES \$(2,3,4,\'$dbdate\',\'$dbtime\',
\'BDF:$filesys\', \'$ipaddr\', \'/usr/bin/df -k \',\'BDF\')\";\""
  done
}

```

Figure 2: Collecting NFS file system data using the df tool.

continuously changing parameters that must be collected at intervals. In our environment, we collect these statistics every five minutes to provide an overview of the system's activity.

The first example, figure 2, acquires details from df, the filesystem disk space reporting facility. This tool will provide details as to the usage of all file systems, including mounted file systems. Significant changes in disk space usage can be identified and reported as a possible attack. More importantly, correlating the results of df over time will show if a network mount is lost for a period of time. This can be an indication of a network problem, a typical system problem, or the subversion of a system.

The code in figure 2 shows the access to the df facility. We first check the parameters accepted by the df command such that we can acquire the appropriate data values on a wide range of systems. The df access tool itself retrieves the name of each file system and the percentage of disk space usage for that file system. The df tool itself will not report when an NFS mount is lost so we must monitor the results of the df tool over time and identify changes in the output to determine when a mount is lost.

The code example in figure 3 uses the who facility to determine the number of users logged into the system at a particular point in time. The benefit of this tool arises in correlation with other tools as it can identify when more than one user is logged onto a system that isn't expecting such or show inconsistencies in the local log files if they do not show a match for the number of users expected to be logged in at any point in time. Since attackers will compromise the *standard* log reporting facilities as one of their principal tasks, this mechanism provides an ability to identify inconsistencies in the log reporting mechanism. This also shows the value of a separate database system for the recording of log information.

The final example of the section, figure 4, uses the uptime tool to report the system usage in terms of system load. While a very active server will always have a continuous load with localized peaks, if the system load should be high for an extended period of time then this is an indication that something unexpected may be occurring. This can range from runaway programs that need to be removed from the system to an active attack. A typical scenario is that of a naive student running a password cracker on a heavily monitored system.

```

function WhoTool {
usrCnt=$(who -q | awk -F= '/users/ { print $2;}')
eval $PSQL -c "\"INSERT INTO log VALUES \$(2,3,4,\'$dbdate\',\'$dbtime\',
\'USERS:$usrCnt\',\'$ipaddr\', \'who -q\',\'Who\')\";\""
}

```

Figure 3: Monitoring user activity with the who command.

```

function UptimeTool {
cpuload=$(uptime | uptime|sed -e "s/.*average: //g" | tr -s [\ ] | awk '{
print $1" "$1" "$3;}'|tr -d [,])
eval $PSQL -c \"INSERT INTO log VALUES \$(2,3,4,\"'$dbdate'\',\"'$dbtime'\',
\`LOAD:$cpuload\`,`'$ipaddr'\`,`'$uptime'\`,`'$Uptime'\`)\`;\`\"
}

```

Figure 4: Collecting system usage statistics with the uptime tool.

3.3. System Last Record

Additional information is collected from the system last record. The system last record reports user connection and disconnection time, after authentication. The system last record is also event based as with the tools described in section 3.1. The user connection and disconnection times aids in identifying anomalous behavior as far as user history goes. Correlating user activity with nodes previously identified to have been probing the system or failing authentications repeatedly can be an indication of a successful attack or misuse. As another example, students are not suppose to share their login information with anyone else. Should a student be connected to the system from two completely different locations at the same time then misuse is indicated.

The last record is accessed using a command called *last*. We were forced to modify the source code of last, and subsequently its behavior, as the typical last command does not display the entire remote host name, rather only displaying the first 15 characters. Retrieving the entire hostname is crucial if we are to effectively correlate activities of one system over time and prevent name space collisions. The tool must be written in C since the last record is a database in binary format and the

structure for accessing the log is described through C header files, namely utmp.h.

The Bourne script segment shown in figure 5 parses the textual data reported by the modified last command, identifies whether an event is a connection or disconnection, converts textual months to integer representations, and finally posts the event to the postgres server. The tool converts data output from the modified last tool to a format consistent with Hummer.

3.4. Network Traffic

Network traffic collected from a typical network sniffer provides the last of our data sources. This data is the most demanding due to its sheer volume. Consequently, statistical techniques are generally required to sample the network activity. This can miss information on the network but should capture ongoing activity, even if intermittent. Capturing network traffic data will often lead to surprising results as to what activity is occurring on the network and has frequently been used to identify intrusions. Each packet on the network, whether TCP/IP or UDP based, provides header information identifying the type of data being transmitted, the originating system, and the destination system. The type of data being transmitted will directly implicate the tool used to generate the data. This has been used

```

function LastTool {
tail +"$LAST_COUNT"1 /usr/tmp/logins.singer.$lhost >/usr/tmp/tmpa.`hostname`
LAST_COUNT=`wc -l /usr/tmp/logins.singer.$lhost | tr -s [:space:]|cut -d\ -f
2`
while read flag uid dev host wday mon day start
do
[ $flag = "c" ] && FROMTO="in on"
[ $flag = "d" ] && FROMTO="out of"
monno=$(echo -e "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug\nSep\nOct\nNov\n
Dec\n" | sed -n '/'$mon'/=' )
dbdate="$monno-$day-$(date +%Y)"
dbtime="$start"
message="User $uid logged $FROMTO $dev from $host"
eval $PSQL -c \"INSERT INTO log VALUES \$(2,3,4,\"'$dbdate'\',\"'$dbtime'\',
\`SUNY: $message\`,`'$ipaddr'\`,`'$wtmpfile'\`,`'$System'\`)\`;\`\"
done </usr/tmp/tmpa.`hostname`
}

```

Figure 5: Script for collecting data from the modified last command.

numerous times to identify illegal IRC servers, most often being run on compromised systems. It is also used to identify systems illegally connected to the local subnetwork and compromised systems no longer providing complete or correct log facilities.

4. Future Work

While we have provided extensive capabilities for the monitoring of system events there remain many issues related to performance and security. In particular, scripts are not efficient in terms of CPU usage, so while the script is efficient to write it consumes an enormous amount of CPU resources. In conjunction with this we are exploring a C-based port of the data collection facilities provided by the script. Second, we must examine the network bandwidth consumed by the data collection and examine the applicability of data compression before sending the events over the network. This will result in a comparison of the network bandwidth of the environment vs. the CPU usage of the environment.

Finally, we need to examine the applicability of encryption to prevent the compromise of data sent over the network. While none of the data we are collecting is particularly sensitive and is in essence available to all users on the system, should an attacker identify the data being sent over the network it may inform them of additional tasks they need to perform in order to ensure their activity cannot be traced. Again, the performance impact of the encryption algorithms must be measured.

5. Conclusions

We have described the data collection facilities we have developed within the environs of a Bourne script. The data collected by such a script provides enormous capabilities to further the analysis of system information. Currently, log files provide the main avenue for system administrators to monitor systems for an attack [10]. This, however, is totally inadequate and the situation is only going to get worse in the future as ever more hackers attempt to intrude on systems and the number of events system administrators must analyze on a system increases. Publishing

the details presented in this paper is the first step in assisting other groups in collecting data and developing advanced analysis tools. For details on our visualization which assists in correlating and analyzing the data described in this paper please review [5].

6. Acknowledgments

We would like to acknowledge Ron Goebel for his assistance in the granting of permission for the scripts to be run unattended for lengthy durations and the initialization of configuration values conducive to the data collection.

References

- [1] D. Rae and D. Ludlow, "Halt! Who goes there? [Internet intrusion detection benchtest]," *Network News (UK Edition)*, February 16, 2000, pp. 31-37.
- [2] <http://www.psionic.com/products/>.
- [3] G.H. Kim, E.H. Spafford, "Writing, supporting, and evaluating Tripwire: a publicly available security tool," *Proceedings of the 1994 USENIX. UNIX Applications Development Symposium*, April 1994, pp. 89-107.
- [4] J.A. Fore, "System security: when enough is not enough," *Proceedings of Expanding Expectations in Integrated Online Library Systems (IOLS)*, New York, 1997, pp. 53-62.
- [5] Robert F. Erbacher, Kenneth L. Walker, and Deborah A. Frincke, "Intrusion and Misuse Detection in Large-Scale Systems," *Computer Graphics and Applications*, Vol. 22, No. 1, January/February 2002, pp. 38-48.
- [6] Polla, D., J. McConnell, T. Johnson, J. Marconi, D. Tobin, and D. Frincke, "A Framework for Cooperative Intrusion Detection," *21st National Information Systems Security Conference*, pp. 361-373, October 1998.
- [7] S. Baker, "LDAP: the glue is almost set," *UNIX Review's Performance Computing*, Vol. 16, No. 12, November 1998, pp. 31-38.
- [8] <http://man.he.net/man8/inetd>
- [9] J. Nazario, "Using xinetd [Linux]," *Linux Journal*, No. 83, March 2001, pp. 136-141.
- [10] Rebecca Gurley Bace, *Intrusion Detection*, Macmillan Technical Publishing, 2000.