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

Neil Desai
GCFA ver 1.0

Part I

Scope

I will be reviewing a binary parsing utility called *strings*. The test will be conducted in a protected environment so that outside interference can be kept to a minimum. The purpose of this test is to show the usefulness of this utility and also show what can be expected in terms of information gathering. This utility will be used on some UNIX C programs that I will write so that we can see not only the source code, but also the information that is gathered. The tests conducted will also show that the evidence that is obtained is verifiable and repeatable. The *strings* utility is available on both UNIX and Windows. While each version of *strings* may have it's quirks they all still essentially show the same information. It may just be a difference of the options used when running the *strings* utility. We will be running *strings –a*. This option allows us to look at the entire binary.

The tests conducted will be limited to an ELF (Executable and Linkable Format) formatted binary on a Redhat Linux system. The reason that we are limiting the way that the binary is created is that different binaries formats will produce different results. We will be able to look at documentation on the ELF format and other UNIX/LINUX documentation to support our findings on the output from *strings*.

Tools Description

The program that I will be examining will be */usr/bin/strings* (NOTE: For brevity the */usr/bin/strings* utility will be referenced as *strings* for the rest of the paper regarding this utility). This program is part of the *binutils* collection of binary utilities. The *strings* utility is used to see any printable text within a binary file. Like most UNIX utilities there are many options that can be used with it, but we will only be using the *–a* option. This will allow the forensic analyst to examine the entire binary file. Below is an excerpt from the *strings* man page.

> strings - print the strings of printable characters in files.
> ……..
> `-a'
> `--all'
> `-' Do not scan only the initialized and loaded sections
> of object files; scan the whole files.

The *strings* utility will usually be one of the first utilities used in reverse engineering a file. The file does not have to be a binary though, it can be used on any file that at a glance does not have any printable text. I will demonstrate the

usefulness of *strings* in forensic analysis/reverse engineering on binaries that I will create so that we can see the source code and compare it to theinformation we are expecting to get. The version of *strings* is 2.11.90.0.8 as seen below.

```
[root@localhost binary]# strings -v
GNU strings 2.11.90.0.8
Copyright 1997, 98, 99, 2000, 2001 Free Software Foundation, Inc. This program is free
software; you may redistribute it under the terms of the GNU General Public License.
This program has absolutely no warranty.
```

This is the default version of strings in Redhat 7.2. The source RPM can be downloaded from Redhat's FTP site (ftp://ftp.redhat.com/pub/redhat/linux/updates/7.2/en/os/i386/SRPMS/binutils-2.11.90.0.8-12.src.rpm). The *strings* utility is used to see any printable text that exists within a file. This will give the analyst a peek into the file to get any clues as to what the file is and/or does.

When dealing with an unknown file the forensic analyst must proceed with caution, even in a "protected environment" (i.e. VMWare in host-only mode). Usually the first utility that will be run is */usr/bin/file*. This utility tells the analyst the type of file that is being looked at. This utility can be fooled and give false information. A discussion of this is outside the scope of the file (*strings*) being discussed, so we will assume that we are getting correct information from */usr/bin/file*. The file type will determine what steps the analyst takes to deal with the file.

If the file were to show up as some type of binary the analyst would need to get some clues about the binary. Depending on the skill level of the analyst he/she may take a different approach to analyzing the binary. One approach would be to load the binary into a disassembler (i.e. IDA Pro, */usr/bin/objdump*) and look at the assembly code output. This requires a lot of skill to reassemble the binary, so most analysts will opt to use *strings* to gain information about the binary before actually disassembling the binary.

Strings does use some outside libraries if it is not statically compiled. This can be seen from the output of */usr/bin/ldd*. The reason that we used */usr/bin/ldd* is so that we could see not only the direct dependencies, but also the indirect dependencies. We can see that *strings* uses the following shared libraries: */usr/lib/libbfd-2.11.90.0.8.so, /lib/i686/libc.so.6, /lib/libdl.so.2*, */lib/ld-linux.so.2* (indirectly). We also see that */usr/lib/libbfd-2.11.90.0.8.so* and */lib/libdl.so.2* are dependent on */lib/i686/libc.so.6.* We see that */lib/i686/libc.so.6* is dependent on */lib/ld-linux.so.2.*

```
[root@localhost tools]# /usr/bin/ldd -v /usr/bin/strings
libbfd-2.11.90.0.8.so => /usr/lib/libbfd-2.11.90.0.8.so
(0x4001e000)
libdl.so.2 => /usr/lib/libbfd-2.11.90.0.8.so (0x40077000)
libc.so.6 => /lib/i686/libc.so.6 (0x4007b000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

```
        Version information:
/usr/bin/strings:
   libc.so.6 (GLIBC_2.1.3) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.1) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.0) => /lib/i686/libc.so.6
/usr/lib/libbfd-2.11.90.0.8.so:
   libc.so.6 (GLIBC_2.1.3) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.2.3) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.1) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.0) => /lib/i686/libc.so.6
/lib/libdl.so.2:
   libc.so.6 (GLIBC_2.1.3) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.1) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.2) => /lib/i686/libc.so.6
   libc.so.6 (GLIBC_2.0) => /lib/i686/libc.so.6
/lib/i686/libc.so.6:
   ld-linux.so.2 (GLIBC_2.1.1) => /lib/ld-linux.so.2
   ld-linux.so.2 (GLIBC_2.2.3) => /lib/ld-linux.so.2
   ld-linux.so.2 (GLIBC_2.1) => /lib/ld-linux.so.2
   ld-linux.so.2 (GLIBC_2.2) => /lib/ld-linux.so.2
   ld-linux.so.2 (GLIBC_2.0) => /lib/ld-linux.so.2
```

To document what these files are we will start by looking at the naming structure for these files. Shared object files have two names, the *soname* (Shared Object name) and the *real name*. The files will usually be prefixed with *lib* followed by the name of the library, then the "*so.*" (note the dot at the end of *so*) and then the major version number. The *real name* is the actual file name that contains the compiled code for the library. Now that we know the naming convention for the files we will find out what each one of them does and/or is. The file *libc.so.6* is a symbolic link to *libc-2.2.4.so*. The file *libc-2.2.4.so* is the C shared library for the 2.2.4 kernel. The file *ld-linux.so.2* is a symbolic link to *ld-2.2.4.so*. The file *ld-2.2.4.so* is a run time link editor. The file *libbfd-2.11.90.0.8.so* is the library for the "Binary File Descriptor" version 2.11.90.0.8. The *bfd* provides a common interface to an object file. The file *libdl.so.2* is a symbolic link to *libdl-2.2.4.so*. The file *libdl-2.2.4.so* is used for dynamic linking. I was not able to find more information about the file *libdl-2.2.4.so*. *Strings* can be compiled statically so that it can be run from a floppy or CDROM without interacting with these operating systems shared object libraries. This has been done on different versions of *strings* for different distributions of forensic toolkits.

Test Apparatus

To analyze the *strings* utility we will use the same system that we used for the second part of the practical:

1. Windows 2000 SP1: This workstation has ActiveState's PERL 5.6.0, UltraEdit32 7.10a, 3Cdaemon v2/rev10 (FTP server), SecureCRT 3.3.2, VMWare 3.1.1 build 1790 and Cygwin 1.3.10 installed on it.
2. Redhat 7.2 (VMWare, host-only mode): I choose to install everything. *Mac_daddy.pl* was also installed on the system. *Mac_daddy.pl* is a PERL script written by Rob Lee (http://www.incident-response.org/mac_daddy.html) which will show a user the MACTimes of files and directories.

The reason that I choose to install Redhat in a VMWare session is because VMWare is a virtual machine this offers me the ability to use a second system for the analysis without the cost of a second system. U.S. government agencies have looked into VMWare because of it's security in providing multiple virtual machines within a single physical machine. When VMWare is used in a host-only mode for network connectivity the VMWare session is only able to connect to the host machine. The IP addresses that are assigned are part of the 192.168.x.x private address space. The host machine also has a second interface (virtual) that connects to the VMWare session. This way even if the host is connected to another network the VMWare session can only talk with the host machine. We will connect to the VMWare session with SecureCRT 3.3.2 using the SSHv2 protocol. SecureCRT is a client application that supports terminal emulation through SSH. SecureCRT provides us with logging capabilities so that we can log all commands and output from the commands.

Environmental Conditions

The tool was tested in a VMWare session of Redhat Linux 7.2. The VMWare session was set up to use host-only mode for the networking configuration. This was the safest way to reverse engineer the binary. The binary has no connection to the Internet and at worst it destroys the VMWare session. Since the VMWare session is setup in host-only mode the only device that can interact with the Linux workstation would be the host that the VMWare session is running in.

Description of the Procedures

As noted in the previous section we will be testing *strings* on a RedHat Linux 7.2 server that was listed in out "Test Apparatus" section of the practical. The RedHat server will be accessed via SSH2 using SecureCRT on the host workstation. All the output will be captured to a local log file and then copied to this document.

To see what the *strings* utility will show about a binary I will be writing some simple programs and compiling them. This way we know what is in the

source code and what information we should look for in the output from *strings –a*. To limit the outside influence of other utilities, the binaries will not be stripped or altered in any way after the compilation process. Because of this we will get a lot of information, most of which is useless to us.

To make sure that anyone wanting to repeat this process can do so I will provide the version numbers of the software and the MD5 checksums. Normally the version number would be sufficient, but since this is a forensic validation of a tool I will use */usr/bin/md5sum* to get the MD5 checksum of the binary. We will run all the binaries using the full path to the tool or utility. This will ensure that the correct binary is run.

We are going to use */usr/bin/gcc* version *2.96 20000731 (Red Hat Linux 7.1 2.96-98)* to compile all the C source code.

```
[root@localhost mac_daddy]# /usr/bin/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)
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/bin/gcc
f5c6d63e2d510d47138dbd90ea0c2591  /usr/bin/gcc
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/lib/gcc-lib/i386-redhat-
linux/2.96/specs
922b60125245fb15b076d39e52ac6616  /usr/lib/gcc-lib/i386-redhat-linux/2.96/specs
[root@localhost mac_daddy]#
```

Since */usr/bin/gcc* gets it's version information from */usr/lib/gcc-lib/i386-redhat-linux/2.96/specs* I documented the md5checksum's of the two files. The version of strings is 2.11.90.0.8.

```
[root@localhost mac_daddy]# /usr/bin/strings -v
GNU strings 2.11.90.0.8
Copyright 1997, 98, 99, 2000, 2001 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/bin/strings
bd0918cb5176465ff833294d66c4815a  /usr/bin/strings
[root@localhost mac_daddy]#
```

We will be using the *bash* shell that comes with Redhat 7.2. This version of the *bash* shell is 2.05.8(1).

```
[root@localhost mac_daddy]# /bin/bash --version
GNU bash, version 2.05.8(1)-release (i386-redhat-linux-gnu)
Copyright 2000 Free Software Foundation, Inc.
[root@localhost mac_daddy]# /usr/bin/md5sum /bin/bash
d8868bcb4d60e19ba915c87b27757574  /bin/bash
[root@localhost mac_daddy]#
```

The C source code will be edited with */usr/bin/vim* version 5.8.7.

```
[root@localhost root]# /usr/binvim --version
```

```
VIM - Vi IMproved 5.8 (2001 May 31, compiled Aug  7 2001 10:32:34)
Included patches: 2-7
Compiled by bhcompile@stripples.devel.redhat.com, with (+) or without (-):
+autocmd -browse ++builtin_terms +byte_offset +cindent +cmdline_compl
+cmdline_info +comments +cryptv +cscope +dialog_con +digraphs +emacs_tags +eval
+ex_extra +extra_search +farsi +file_in_path -osfiletype +find_in_path +fork()
-GUI -hangul_input +insert_expand +langmap +linebreak +lispindent +menu
+mksession +modify_fname +mouse +mouse_dec +mouse_gpm +mouse_netterm
+mouse_xterm +multi_byte +perl +python +quickfix +rightleft +scrollbind
+smartindent -sniff +statusline +syntax +tag_binary +tag_old_static
-tag_any_white -tcl +terminfo +textobjects +title +user_commands +visualextra
+viminfo +wildignore +wildmenu +writebackup -X11 -xfontset -xim
-xterm_clipboard -xterm_save
   system vimrc file: "/usr/share/vim/vim58/macros/vimrc"
     user vimrc file: "$HOME/.vimrc"
      user exrc file: "$HOME/.exrc"
  fall-back for $VIM: "/usr/share/vim"
Compilation: gcc -c -I. -Iproto -DHAVE_CONFIG_H     -O2 -march=i386 -mcpu=i686 -
Wall  -fno-strict-aliasing -I/usr/local/include  -I/usr/lib/perl5/5.6.0/i386-linux/CORE  -
I/usr/include/python1.5
Linking: gcc -rdynamic   -o vim         -lncurses -lgpm  -rdynamic  -L/usr/local/lib
/usr/lib/perl5/5.6.0/i386-linux/auto/DynaLoader/DynaLoader.a -L/usr/lib/perl5/5.6.0/i386-
linux/CORE -lperl -ldl -lm -lcrypt /usr/lib/python1.5/config/libpython1.5.a -lieee -ldl -
lpthread -lm
[root@localhost root]# /usr/bin/md5sum /usr/bin/vim
4a5bb867c8fbec274cb90fb0ba7c7a2f  /usr/bin/vim
[root@localhost root]#
```

I will use the command */usr/bin/vim test.c* to start the editing of the files and I will save the files by using the commands "*[ESC]:wq!*".

Each time a piece of source code is written I will use the */bin/cat* command to show the source code. This is easier than attempting to show the output from */usr/bin/vim*. The version of */bin/cat* that we will use is 2.0.14.

```
[root@localhost mac_daddy]# /bin/cat --version
cat (textutils) 2.0.14
Written by Torbjorn Granlund and Richard M. Stallman.

Copyright (C) 2001 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
[root@localhost mac_daddy]# /usr/bin/md5sum /bin/cat
30bef954ee5b8df11101aa1d7ba531fd  /bin/cat
[root@localhost mac_daddy]#
```

I will also use the */usr/bin/objdump* utility. This will allow us to disassemble the binary and look at specific parts of the binary.

```
[root@localhost tools]# /usr/bin/objdump -V
```

```
GNU objdump 2.11.90.0.8
Copyright 1997, 98, 99, 2000, 2001 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
[root@localhost tools]# /usr/bin/md5sum /usr/bin/objdump
c32907f70f89eab4bcbd48aa4692907e  /usr/bin/objdump
[root@localhost tools]#
```

I will first write a simple "Hello World" program. This will allow us to look at one of the most basic programs. From there I will edit the *test.c* C source code file and add other pieces of information that are commonly found in programs. We will limit our testing to following items:

1. Basic *printf* function.
2. Comments within the C source code.
3. The *#define* preprocessor.
4. Global variables, integer arrays, and character arrays.
5. Local variables, integer arrays, and character arrays.
6. The process of taking a C source code file and making it an executable.
7. What the C source code looks like when converted to assembly language.

To see everything that strings can find within a binary we would have to look at all the system calls and functions available to the C language in the Linux environment. This task would take a few months to complete and is outside the scope of this project.

I have come up with a process for forensic analysts to find things that I have not directly discussed in regards to the information gathered by *strings*. By looking at the assembly output of an ELF binary (*objdump –D elfbinaryfile* (where elfbinaryfile is the ELF binary file that the analyst wants to disassemble)) we can see the system calls being made (int $0x80). By looking at the value in the *eax* register just before the *int $0x80* system call and cross referencing it will the information in */usr/include/asm/unistd.h* we can find out what system calls are being made. We can then write small pieces of code using those system calls can examine them with the *strings* utility to find out what information about the system calls we can find with the *strings* utility.

Before I start writing the test programs we need some background information on the binary compiling process and the tools that we will be using to document and verify the information. When writing a program we will start out with a basic ACSII text file containing instructions formatted for the C programming language. The C programming language is considered a high level language. We will save our C source code as files that will have the ".c" file extension (minus the quotes). When *gcc* sees a file with this extension it assumes that the file is a C source code file.

Below I list the steps that the C source code will take to be compiled into a ELF formatted binary.

1. Start out with the C source code (ASCII Text file).
2. The C source code will be fed to the compiler, *gcc*.
3. The compiler will then turn the C source code into assembly language.
4. The assembler, *as*, will take the assembly language and create the object code.
5. The linker, *ld-2.2.4.so*, will take the object code and create the final binary.

By default when running the */usr/bin/gcc* complier it will take the C source code and compile it into a ELF formatted binary. There are options for */usr/bin/gcc* so that we can create just the assembly language of a C program (-S option) or create an object file (-c option). From the assembly language file or the object file we can use */usr/bin/as* or */usr/bin/ld*, respectively, to create the final binary. We will need to understand the layout of the ELF binary so that we know what information we can get and from where we will be getting it.

The main thing that we need to understand is that certain data will only be seen when the executable is ran. To see what the binary stores we will look at the output from */usr/bin/objdump* (NOTE: For the rest of the presentation we will refer to */usr/bin/objdump* as *objdump*). We will assume that *objdump* has all ready been proven forensically sound so that we can rely on the output from it.

The ELF binary is made up of a number of different sections used by the system[i]. The most useful sections for us will be *.comment*, *.rodata*, *.data*, and *.text*. The *.comment* section will contain version control information. The *.rodata* section will contain read-only information. The *.data* section will contain initialized data. The *.text* section will contain the actual program code.

Criteria for Approval

The *strings* utility should show us printable strings within binary files as well as parts of the source code and information about what was used to compile it (i.e. /usr/bin/gcc, cc), operating system it was compiled on (Redhat Linux, Mandrake Linux) and what libraries are a part of the binary. From the output that we will show we will be able see that the information is repeatable and verifiable.

The *strings* utility will be executed from the command line or shell. On a Unix/Linux system the forensic analyst will execute *strings* via the full path, */usr/bin/strings*. For forensic purposes all command will be executed via the full path to ensure that the correct binary is run.

The *strings* utility does use some system files as can be seen below:

[i] Executable and Linkable Format (ELF) version 1.1 page 14

```
[root@localhost tools]# /usr/bin/ldd  /usr/bin/strings
        libbfd-2.11.90.0.8.so => /usr/lib/libbfd-2.11.90.0.8.so
        (0x4001e000)
        libdl.so.2 => /lib/libdl.so.2 (0x40077000)
        libc.so.6 => /lib/i686/libc.so.6 (0x4007b000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

More detailed information about the dependencies are contained in the "Tools Description" portion of the forensic tool validation part of the practical. If the *strings* utility has write access to the file being examined then the atime of the binary would change. To show this we are going to use the Linux */bin/date* command and *mac_daddy.pl* (a PERL script, written by Rob Lee, for showing the MACTimes of files). First we will compile a binary with */usr/bin/gcc.* The binaries MACTimes should all be the same. We will confirm this with *mac_daddy.pl*. The *strings* utility does not alter the actual binary. We will prove this will the help of */usr/bin/md5sum.* The test will be conducted on a regular partition so that we can see if the *strings* utility alters a file's metadata or content.
The first test we will conduct is to prove that *strings* only changes the atime of the binary.

```
[root@localhost mac_daddy]# /bin/date
Tue Oct 15 08:48:09 EDT 2002
[root@localhost mac_daddy]# /usr/bin/gcc -o/usr/tools/tools/test /usr/tools/tools/test.c
[root@localhost mac_daddy]# /bin/date
Tue Oct 15 08:48:27 EDT 2002
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/tools/
[root@localhost mac_daddy]# Oct 15 2002 05:58:38     1475 m.. -rw-r--r-- root     root
/usr/tools/tools/test.c
Oct 15 2002 07:53:55     1475 ..c -rw-r--r-- root     root     /usr/tools/tools/test.c
Oct 15 2002 08:48:25    14520 mac -rwxr-xr-x root     root     /usr/tools/tools/test
                  1475 .a. -rw-r--r-- root     root     /usr/tools/tools/test.c

 [root@localhost mac_daddy]# /usr/bin/strings -a /usr/tools/tools/test >output
 [root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/tools/
[root@localhost mac_daddy]# Oct 15 2002 05:58:38     1475 m.. -rw-r--r-- root     root
/usr/tools/tools/test.c
Oct 15 2002 07:53:55     1475 ..c -rw-r--r-- root     root     /usr/tools/tools/test.c
Oct 15 2002 08:48:25    14520 m.c -rwxr-xr-x root     root     /usr/tools/tools/test
                  1475 .a. -rw-r--r-- root     root     /usr/tools/tools/test.c
Oct 15 2002 08:49:10    14520 .a. -rwxr-xr-x root     root     /usr/tools/tools/test
```

From the above output we can clearly see that *strings* only changes the atime of the binary. The reason that the atime is changes is because *strings* has to access/read the file in order to find the printable characters. When the file is read the atime changes.

Next we will check the integrity of the binary before and after the *strings* utility is run on it.

```
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/tools/tools/test
```

```
1a80f714206216c542917d1591a440c9  /usr/tools/tools/test
[root@localhost mac_daddy]# /usr/binstrings -a /usr/tools/tools/test >ouput
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/tools/tools/test
1a80f714206216c542917d1591a440c9  /usr/tools/tools/test
[root@localhost mac_daddy]
```

From the above information we can see that the *strings* utility does not change any of the data within the binary.

Even though the *strings* utility changes the atime of a file it is still a very valuable tool in a forensic investigation. The *strings* utility will rarely be used on a live system. Most of the time it will be used on an image of the hard drive. In this case the forensic analyst would mount the hard drive as read only. This can be done within the OS or with special hardware. Since the information will be read only *strings* will not be able to change the atime of the binary.

The *strings* utility will provide the forensic analyst a high level overview of the binary. The information gathered during this analysis could save the forensic analyst a lot of time and reverse engineering as we will see in the next section.

If the file is encrypted then there might not be much useful information gathered by *strings,* but currently there are not many attackers encrypting their binaries. Most attacks will just *strip* the binaries to lessen the footprint, which also makes it harder to disassemble. The stripping process does not make it impossible for someone to gain valuable information via *strings*, just harder to reverse engineer.

Data and Results

The first program that I will write is a simple executable that prints “Hello world” to the screen. I will then compile it with */usr/bin/gcc* and then use *strings -a* on the complied binary.

```
[root@localhost tools]# /bin/cat test.c

int main()
{
printf("Hello world\n");
return(0);
}

[root@localhost tools]# /usr/bin/gcc -o test test.c
[root@localhost tools]# /usr/bin/strings -a test
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
printf
__cxa_finalize
__deregister_frame_info
_IO_stdin_used
__libc_start_main
```

```
__register_frame_info
GLIBC_2.1.3
GLIBC_2.0
ÉÃÿ5
éàÿÿÿÿ%,
éÐÿÿÿÿ%0
éÀÿÿÿÿ%4
é°ÿÿÿÿ%8
é ÿÿÿ
äðPTRhÀ
QVh`
è«ÿÿÿô
åSPè
]üÉÃ
Éuê¸
è ÿÿÿ
èËþÿÿ
èÉþÿÿ
øÿuôX[]ÃU
åSRè
è×þÿÿ
]üÉÃ
Hello world
ÿÿÿÿ
ÿÿÿÿ
þÿÿo
ÿÿÿo
ðÿÿor
init.c
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/csu/
/usr/bin/gcc2_compiled.
int:t(0,1)=r(0,1);-2147483648;2147483647;
char:t(0,2)=r(0,2);0;127;
long int:t(0,3)=r(0,3);-2147483648;2147483647;
unsigned int:t(0,4)=r(0,4);0000000000000;0037777777777;
long unsigned int:t(0,5)=r(0,5);0000000000000;0037777777777;
long long
int:t(0,6)=@s64;r(0,6);01000000000000000000000;07777777777777777777;
long long unsigned int:t(0,7)=@s64;r(0,7);0000000000000;01777777777777777777777;
short int:t(0,8)=@s16;r(0,8);-32768;32767;
short unsigned int:t(0,9)=@s16;r(0,9);0;65535;
signed char:t(0,10)=@s8;r(0,10);-128;127;
unsigned char:t(0,11)=@s8;r(0,11);0;255;
float:t(0,12)=r(0,1);4;0;
double:t(0,13)=r(0,1);8;0;
long double:t(0,14)=r(0,1);12;0;
complex int:t(0,15)=s8real:(0,1),0,32;imag:(0,1),32,32;;
complex float:t(0,16)=r(0,16);8;0;
complex double:t(0,17)=r(0,17);16;0;
complex long double:t(0,18)=r(0,18);24;0;
__builtin_va_list:t(0,19)=*(0,20)=(0,20)
../include/libc-symbols.h
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/build-i386-linux/config.h
```

```
../sysdeps/gnu/_G_config.h
../sysdeps/unix/sysv/linux/bits/types.h
../include/features.h
../include/sys/cdefs.h
../misc/sys/cdefs.h
/usr/lib//usr/bin/gcc-lib/i386-redhat-linux/2.96/include/stddef.h
size_t:t(8,1)=(0,4)
__u_char:t(4,1)=(0,11)
__u_short:t(4,2)=(0,9)
__u_int:t(4,3)=(0,4)
__u_long:t(4,4)=(0,5)
__u_quad_t:t(4,5)=(0,7)
__quad_t:t(4,6)=(0,6)
__int8_t:t(4,7)=(0,10)
__uint8_t:t(4,8)=(0,11)
__int16_t:t(4,9)=(0,8)
__uint16_t:t(4,10)=(0,9)
__int32_t:t(4,11)=(0,1)
__uint32_t:t(4,12)=(0,4)
__int64_t:t(4,13)=(0,6)
__uint64_t:t(4,14)=(0,7)
__qaddr_t:t(4,15)=(4,16)=*(4,6)
__dev_t:t(4,17)=(4,5)
__uid_t:t(4,18)=(4,3)
__gid_t:t(4,19)=(4,3)
__ino_t:t(4,20)=(4,4)
__mode_t:t(4,21)=(4,3)
__nlink_t:t(4,22)=(4,3)
__off_t:t(4,23)=(0,3)
__loff_t:t(4,24)=(4,6)
__pid_t:t(4,25)=(0,1)
__ssize_t:t(4,26)=(0,1)
__rlim_t:t(4,27)=(4,4)
__rlim64_t:t(4,28)=(4,5)
__id_t:t(4,29)=(4,3)
__fsid_t:t(4,30)=(4,31)=s8__val:(4,32)=ar(4,33)=r(4,33);0000000000000;0037777777777
;;0;1;(0,1),0,64;;
__daddr_t:t(4,34)=(0,1)
__caddr_t:t(4,35)=(4,36)=*(0,2)
__time_t:t(4,37)=(0,3)
__useconds_t:t(4,38)=(0,4)
__suseconds_t:t(4,39)=(0,3)
__swblk_t:t(4,40)=(0,3)
__clock_t:t(4,41)=(0,3)
__clockid_t:t(4,42)=(0,1)
__timer_t:t(4,43)=(0,1)
__key_t:t(4,44)=(0,1)
__ipc_pid_t:t(4,45)=(0,9)
__blksize_t:t(4,46)=(0,3)
__blkcnt_t:t(4,47)=(0,3)
__blkcnt64_t:t(4,48)=(4,6)
__fsblkcnt_t:t(4,49)=(4,4)
__fsblkcnt64_t:t(4,50)=(4,5)
```

```
__fsfilcnt_t:t(4,51)=(4,4)
__fsfilcnt64_t:t(4,52)=(4,5)
__ino64_t:t(4,53)=(4,5)
__off64_t:t(4,54)=(4,24)
__t_scalar_t:t(4,55)=(0,3)
__t_uscalar_t:t(4,56)=(0,5)
__intptr_t:t(4,57)=(0,1)
__socklen_t:t(4,58)=(0,4)
../linuxthreads/sysdeps/pthread/bits/pthreadtypes.h
../sysdeps/unix/sysv/linux/bits/sched.h
__sched_param:T(10,1)=s4__sched_priority:(0,1),0,32;;
_pthread_fastlock:T(9,1)=s8__status:(0,3),0,32;__spinlock:(0,1),32,32;;
_pthread_descr:t(9,2)=(9,3)=*(9,4)=xs_pthread_descr_struct:
__pthread_attr_s:T(9,5)=s36__detachstate:(0,1),0,32;__schedpolicy:(0,1),32,32;__sched
param:(10,1),64,32;__inheritsched:(0,1),96,32;_
_scope:(0,1),128,32;__guardsize:(8,1),160,32;__stackaddr_set:(0,1),192,32;__stackaddr
:(0,19),224,32;__stacksize:(8,1),256,32;;
pthread_attr_t:t(9,6)=(9,5)
pthread_cond_t:t(9,7)=(9,8)=s12__c_lock:(9,1),0,64;__c_waiting:(9,2),64,32;;
pthread_condattr_t:t(9,9)=(9,10)=s4__dummy:(0,1),0,32;;
pthread_key_t:t(9,11)=(0,4)
pthread_mutex_t:t(9,12)=(9,13)=s24__m_reserved:(0,1),0,32;__m_count:(0,1),32,32;__
m_owner:(9,2),64,32;__m_kind:(0,1),96,32;__m_lock:
(9,1),128,64;;
pthread_mutexattr_t:t(9,14)=(9,15)=s4__mutexkind:(0,1),0,32;;
pthread_once_t:t(9,16)=(0,1)
_pthread_rwlock_t:T(9,17)=s32__rw_lock:(9,1),0,64;__rw_readers:(0,1),64,32;__rw_writ
er:(9,2),96,32;__rw_read_waiting:(9,2),128,32;__
rw_write_waiting:(9,2),160,32;__rw_kind:(0,1),192,32;__rw_pshared:(0,1),224,32;;
pthread_rwlock_t:t(9,18)=(9,17)
pthread_rwlockattr_t:t(9,19)=(9,20)=s8__lockkind:(0,1),0,32;__pshared:(0,1),32,32;;
pthread_spinlock_t:t(9,21)=(0,1)
pthread_barrier_t:t(9,22)=(9,23)=s20__ba_lock:(9,1),0,64;__ba_required:(0,1),64,32;__b
a_present:(0,1),96,32;__ba_waiting:(9,2),128,3
2;;
pthread_barrierattr_t:t(9,24)=(9,25)=s4__pshared:(0,1),0,32;;
pthread_t:t(9,26)=(0,5)
wchar_t:t(11,1)=(0,3)
wint_t:t(11,2)=(0,4)
../include/wchar.h
../wcsmbs/wchar.h
../sysdeps/unix/sysv/linux/i386/bits/wchar.h
__mbstate_t:t(13,1)=(13,2)=s8__count:(0,1),0,32;__value:(13,3)=u4__wch:(11,2),0,32;__
wchb:(13,4)=ar(4,33);0;3;(0,2),0,32;;,32,32;;
_G_fpos_t:t(3,1)=(3,2)=s12__pos:(4,23),0,32;__state:(13,1),32,64;;
_G_fpos64_t:t(3,3)=(3,4)=s16__pos:(4,54),0,64;__state:(13,1),64,64;;
../include/gconv.h
../iconv/gconv.h
:T(17,1)=e__GCONV_OK:0,__GCONV_NOCONV:1,__GCONV_NODB:2,__GCONV_N
OMEM:3,__GCONV_EMPTY_INPUT:4,__GCONV_FULL_OUTPUT:5,__GCONV_ILLEG
AL_
INPUT:6,__GCONV_INCOMPLETE_INPUT:7,__GCONV_ILLEGAL_DESCRIPTOR:8,__
GCONV_INTERNAL_ERROR:9,;
```

```
 :T(17,2)=e__GCONV_IS_LAST:1,__GCONV_IGNORE_ERRORS:2,;
__gconv_fct:t(17,3)=(17,4)=*(17,5)=f(0,1)
__gconv_init_fct:t(17,6)=(17,7)=*(17,8)=f(0,1)
__gconv_end_fct:t(17,9)=(17,10)=*(17,11)=f(0,20)
__gconv_trans_fct:t(17,12)=(17,13)=*(17,14)=f(0,1)
__gconv_trans_context_fct:t(17,15)=(17,16)=*(17,17)=f(0,1)
__gconv_trans_query_fct:t(17,18)=(17,19)=*(17,20)=f(0,1)
__gconv_trans_init_fct:t(17,21)=(17,22)=*(17,23)=f(0,1)
__gconv_trans_end_fct:t(17,24)=(17,25)=*(17,26)=f(0,20)
__gconv_trans_data:T(17,27)=s20__trans_fct:(17,12),0,32;__trans_context_fct:(17,15),3
2,32;__trans_end_fct:(17,24),64,32;__data:(0,19
),96,32;__next:(17,28)=*(17,27),128,32;;
__gconv_step:T(17,29)=s56__shlib_handle:(17,30)=*(17,31)=xs__gconv_loaded_object:
,0,32;__modname:(17,32)=*(0,2),32,32;__counter:(0,1
),64,32;__from_name:(4,36),96,32;__to_name:(4,36),128,32;__fct:(17,3),160,32;__init_fc
t:(17,6),192,32;__end_fct:(17,9),224,32;__min_
needed_from:(0,1),256,32;__max_needed_from:(0,1),288,32;__min_needed_to:(0,1),32
0,32;__max_needed_to:(0,1),352,32;__stateful:(0,1),3
84,32;__data:(0,19),416,32;;
__gconv_step_data:T(17,33)=s36__outbuf:(17,34)=*(0,11),0,32;__outbufend:(17,34),32,
32;__flags:(0,1),64,32;__invocation_counter:(0,1)
,96,32;__internal_use:(0,1),128,32;__statep:(17,35)=*(13,1),160,32;__state:(13,1),192,6
4;__trans:(17,28),256,32;;
__gconv_info:T(17,36)=s8__nsteps:(8,1),0,32;__steps:(17,37)=*(17,29),32,32;__data:(17
,38)=ar(4,33);0;-1;(17,33),64,0;;
__gconv_t:t(17,39)=(17,40)=*(17,36)
_G_iconv_t:t(3,5)=(3,6)=u44__cd:(17,36),0,64;__combined:(3,7)=s44__cd:(17,36),0,64;_
_data:(17,33),64,288;;,0,352;;
_G_int16_t:t(3,8)=(0,8)
_G_int32_t:t(3,9)=(0,1)
_G_uint16_t:t(3,10)=(0,9)
_G_uint32_t:t(3,11)=(0,4)
_IO_stdin_used:G(0,1)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
01.01
01.01
01.01
01.01
01.01
01.01
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.hash
.dynsym
.dynstr
```

.gnu.version
.gnu.version_r
.rel.dyn
.rel.plt
.init
.plt
.text
.fini
.rodata
.data
.eh_frame
.ctors
.dtors
.got
.dynamic
.sbss
.bss
.stab
.stabstr
.comment
.note
ÿÿÿo
þÿÿo
initfini.c
/usr/bin/gcc2_compiled.
call_gmon_start
init.c
crtstuff.c
__DTOR_LIST__
completed.1
__do_global_dtors_aux
__EH_FRAME_BEGIN__
fini_dummy
object.2
frame_dummy
init_dummy
force_to_data
__CTOR_LIST__
__do_global_ctors_aux
__CTOR_END__
__DTOR_END__
__FRAME_END__
test.c
_DYNAMIC
__register_frame_info@@GLIBC_2.0
_fp_hw
_init
__deregister_frame_info@@GLIBC_2.0
_start
__bss_start
main
__libc_start_main@@GLIBC_2.0
data_start

```
printf@@GLIBC_2.0
_fini
__cxa_finalize@@GLIBC_2.1.3
_edata
_GLOBAL_OFFSET_TABLE_
_end
_IO_stdin_used
__data_start
__gmon_start__
```

Because the binary was not stripped before *strings* was run there is a lot of debugging information that is seen by *strings –a*. Since this is a test of the *strings* utility I will not use */usr/bin/strip* to modify the binary to create a smaller binary and lessen the output from *strings –a*.

From the output of the *test.c* C source code file we can see that there is not much to the program. From the output of *stings* we can see that the binary was compiled with */USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)*, */lib/ld-linux.so.2* and *libc.so.6*. We can also see the output (Hello world) from out *printf* statement, *printf* statement itself and the ELF sections (i.e. .comment, .rodata). Using this information we can determine what type of system the binary was compiled on. The string "test.c" also tells us the name of the original C source code.

The compiler information, ELF headers and linker information is part of the binary, original source code file name and *printf* fuction are all part of the final binary.

To see where the compiler version information came from we will use *objdump*. We will run *objdump* with the *–j* and *–s* options. The *–j* option will allow us to view only the section name that we want to look at. From the documentation of the ELF binary format we know that we want to look at the *.comment* section. The *–s* option lets us look at off the header information that we want to look at.

```
[root@localhost tools]# /usr/bin/objdump -j .comment -s test

test:     file format elf32-i386

Contents of section .comment:
 0000 00474343 3a202847 4e552920 322e3936  .GCC: (GNU) 2.96
 0010 20323030 30303733 31202852 65642048   20000731 (Red H
 0020 6174204c 696e7578 20372e31 20322e39  at Linux 7.1 2.9
 0030 362d3937 29000047 43433a20 28474e55  6-97)..GCC: (GNU
 0040 2920322e 39362032 30303030 37333120  ) 2.96 20000731
 0050 28526564 20486174 204c696e 75782037  (Red Hat Linux 7
 0060 2e312032 2e39362d 39372900 00474343  .1 2.96-97)..GCC
 0070 3a202847 4e552920 322e3936 20323030  : (GNU) 2.96 200
 0080 30303733 31202852 65642048 6174204c  00731 (Red Hat L
 0090 696e7578 20372e31 20322e39 362d3938  inux 7.1 2.96-98
 00a0 29000047 43433a20 28474e55 2920322e  )..GCC: (GNU) 2.
```

```
00b0 39362032 30303030 37333120 28526564  96 20000731 (Red
00c0 20486174 204c696e 75782037 2e312032   Hat Linux 7.1 2
00d0 2e39362d 39382900 00474343 3a202847  .96-98)..GCC: (G
00e0 4e552920 322e3936 20323030 30303733  NU) 2.96 2000073
00f0 31202852 65642048 6174204c 696e7578  1 (Red Hat Linux
0100 20372e31 20322e39 362d3938 29000047   7.1 2.96-98)..G
0110 43433a20 28474e55 2920322e 39362032  CC: (GNU) 2.96 2
0120 30303030 37333120 28526564 20486174  0000731 (Red Hat
0130 204c696e 75782037 2e312032 2e39362d   Linux 7.1 2.96-
0140 39372900                             97).
```

From the above output we can see the version of *gcc* that was used to compile the binary. To see the ELF header information we will use *objdump* with the *–h* option. This will print out summary information about the section headers.

```
[root@localhost tools]# /usr/bin/objdump -h test

test:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .interp       00000013  080480f4  080480f4  000000f4  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  08048108  08048108  00000108  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .hash         00000034  08048128  08048128  00000128  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .dynsym       00000080  0804815c  0804815c  0000015c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .dynstr       00000095  080481dc  080481dc  000001dc  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .gnu.version  00000010  08048272  08048272  00000272  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .gnu.version_r 00000030  08048284  08048284  00000284  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .rel.dyn      00000008  080482b4  080482b4  000002b4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .rel.plt      00000028  080482bc  080482bc  000002bc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .init         00000018  080482e4  080482e4  000002e4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 10 .plt          00000060  080482fc  080482fc  000002fc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 11 .text         00000160  08048360  08048360  00000360  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .fini         0000001e  080484c0  080484c0  000004c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .rodata       00000015  080484e0  080484e0  000004e0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 14 .data         00000010  080494f8  080494f8  000004f8  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 15 .eh_frame     00000004  08049508  08049508  00000508  2**2
```

```
                CONTENTS, ALLOC, LOAD, DATA
 16 .ctors      00000008  0804950c  0804950c  0000050c  2**2
                CONTENTS, ALLOC, LOAD, DATA
 17 .dtors      00000008  08049514  08049514  00000514  2**2
                CONTENTS, ALLOC, LOAD, DATA
 18 .got        00000024  0804951c  0804951c  0000051c  2**2
                CONTENTS, ALLOC, LOAD, DATA
 19 .dynamic    000000c8  08049540  08049540  00000540  2**2
                CONTENTS, ALLOC, LOAD, DATA
 20 .sbss       00000000  08049608  08049608  00000608  2**0
                CONTENTS
 21 .bss        00000018  08049608  08049608  00000608  2**2
                ALLOC
 22 .stab       000007a4  00000000  00000000  00000608  2**2
                CONTENTS, READONLY, DEBUGGING
 23 .stabstr    00001983  00000000  00000000  00000dac  2**0
                CONTENTS, READONLY, DEBUGGING
 24 .comment    00000144  00000000  00000000  0000272f  2**0
                CONTENTS, READONLY
 25 .note       00000078  00000000  00000000  00002873  2**0
                CONTENTS, READONLY
```

From the above output we can see the header names that are part of this binary. To find out were the linker information is kept we will look at the interpreter, *.interp*, section of the ELF binary. This will show us the path to the name of the programs interpreter.

```
[root@localhost tools]# /usr/bin/objdump -j .interp -s test

test:     file format elf32-i386

Contents of section .interp:
 80480f4 2f6c6962 2f6c642d 6c696e75 782e736f  /lib/ld-linux.so
 8048104 2e3200                               .2.
[root@localhost tools]#
```

From the above output we can see that */lib/ld-linux.so* is the name of the interpreter for this binary.

Why is there so much output for such a simple program? Simple. Most of the information is added during the linking phase. Most of the information is related to the *libc* library. We will take a look at an the object file that is created from the *test.c* C source code file. An object file is the output of the assembly language after it is compiled by the assembler but before it is linked to any libraries.

```
[root@localhost tools]# /bin/cat test.c

int main()
{
printf("Hello world\n");
```

```
return(0);
}

[root@localhost tools]# /usr/bin/gcc -c test.c
[root@localhost tools]# /usr/bin/strings -a test.o
èüÿÿÿ
01.01
Hello world
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
.symtab
.strtab
.shstrtab
.text
.rel.text
.data
.bss
.note
.rodata
.comment
 test.c
gcc2_compiled.
main
printf
[root@localhost tools]#
```

From the above output we can see how small the actual program is before it is linked to *libc*. We are also able to more clearly see the information that we retrieved from the final binary.

I will add some comments to the *test.c* program and recompile it to see what our output will look like.

```
[root@localhost tools]# /bin/cat test.c

/*This is a test
of the strings
 binary parsing
 utility*/

 int main()
 {
 printf("Hello world\n");
 return(0);
 }

 [root@localhost tools]# /usr/bin/gcc -o test test.c
[root@localhost tools]# /usr/bin/strings -a test
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
printf
__cxa_finalize
```

```
__deregister_frame_info
_IO_stdin_used
__libc_start_main
__register_frame_info
GLIBC_2.1.3
GLIBC_2.0
ÉÃÿ5
éàÿÿÿÿ%,
éÐÿÿÿÿ%0
éÀÿÿÿÿ%4
é°ÿÿÿÿ%8
é ÿÿÿ
äðPTRhÀ
QVh`
è«ÿÿÿô
åSPè
]üÉÃ
Éuê¸
è ÿÿÿ
èËþÿÿ
èÉþÿÿ
øÿuôX[]ÃU
åSRè
è×þÿÿ
]üÉÃ
Hello world
ÿÿÿÿ
ÿÿÿÿ
þÿÿo
ÿÿÿo
ðÿÿor
init.c
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/csu/
/usr/bin/gcc2_compiled.
int:t(0,1)=r(0,1);-2147483648;2147483647;
char:t(0,2)=r(0,2);0;127;
long int:t(0,3)=r(0,3);-2147483648;2147483647;
unsigned int:t(0,4)=r(0,4);0000000000000;0037777777777;
long unsigned int:t(0,5)=r(0,5);0000000000000;0037777777777;
long long int:t(0,6)=@s64;r(0,6);01000000000000000000000;07777777777777777777;
long long unsigned int:t(0,7)=@s64;r(0,7);0000000000000;01777777777777777777777;
short int:t(0,8)=@s16;r(0,8);-32768;32767;
short unsigned int:t(0,9)=@s16;r(0,9);0;65535;
signed char:t(0,10)=@s8;r(0,10);-128;127;
unsigned char:t(0,11)=@s8;r(0,11);0;255;
float:t(0,12)=r(0,1);4;0;
double:t(0,13)=r(0,1);8;0;
long double:t(0,14)=r(0,1);12;0;
complex int:t(0,15)=s8real:(0,1),0,32;imag:(0,1),32,32;;
complex float:t(0,16)=r(0,16);8;0;
complex double:t(0,17)=r(0,17);16;0;
complex long double:t(0,18)=r(0,18);24;0;
__builtin_va_list:t(0,19)=*(0,20)=(0,20)
```

```
../include/libc-symbols.h
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/build-i386-linux/config.h
../sysdeps/gnu/_G_config.h
../sysdeps/unix/sysv/linux/bits/types.h
../include/features.h
../include/sys/cdefs.h
../misc/sys/cdefs.h
/usr/lib//usr/bin/gcc-lib/i386-redhat-linux/2.96/include/stddef.h
size_t:t(8,1)=(0,4)
__u_char:t(4,1)=(0,11)
__u_short:t(4,2)=(0,9)
__u_int:t(4,3)=(0,4)
__u_long:t(4,4)=(0,5)
__u_quad_t:t(4,5)=(0,7)
__quad_t:t(4,6)=(0,6)
__int8_t:t(4,7)=(0,10)
__uint8_t:t(4,8)=(0,11)
__int16_t:t(4,9)=(0,8)
__uint16_t:t(4,10)=(0,9)
__int32_t:t(4,11)=(0,1)
__uint32_t:t(4,12)=(0,4)
__int64_t:t(4,13)=(0,6)
__uint64_t:t(4,14)=(0,7)
__qaddr_t:t(4,15)=(4,16)=*(4,6)
__dev_t:t(4,17)=(4,5)
__uid_t:t(4,18)=(4,3)
__gid_t:t(4,19)=(4,3)
__ino_t:t(4,20)=(4,4)
__mode_t:t(4,21)=(4,3)
__nlink_t:t(4,22)=(4,3)
__off_t:t(4,23)=(0,3)
__loff_t:t(4,24)=(4,6)
__pid_t:t(4,25)=(0,1)
__ssize_t:t(4,26)=(0,1)
__rlim_t:t(4,27)=(4,4)
__rlim64_t:t(4,28)=(4,5)
__id_t:t(4,29)=(4,3)
__fsid_t:t(4,30)=(4,31)=s8__val:(4,32)=ar(4,33)=r(4,33);0000000000000;0037777777777;
;0;1;(0,1),0,64;;
__daddr_t:t(4,34)=(0,1)
__caddr_t:t(4,35)=(4,36)=*(0,2)
__time_t:t(4,37)=(0,3)
__useconds_t:t(4,38)=(0,4)
__suseconds_t:t(4,39)=(0,3)
__swblk_t:t(4,40)=(0,3)
__clock_t:t(4,41)=(0,3)
__clockid_t:t(4,42)=(0,1)
__timer_t:t(4,43)=(0,1)
__key_t:t(4,44)=(0,1)
__ipc_pid_t:t(4,45)=(0,9)
__blksize_t:t(4,46)=(0,3)
__blkcnt_t:t(4,47)=(0,3)
__blkcnt64_t:t(4,48)=(4,6)
```

```
__fsblkcnt_t:t(4,49)=(4,4)
__fsblkcnt64_t:t(4,50)=(4,5)
__fsfilcnt_t:t(4,51)=(4,4)
__fsfilcnt64_t:t(4,52)=(4,5)
__ino64_t:t(4,53)=(4,5)
__off64_t:t(4,54)=(4,24)
__t_scalar_t:t(4,55)=(0,3)
__t_uscalar_t:t(4,56)=(0,5)
__intptr_t:t(4,57)=(0,1)
__socklen_t:t(4,58)=(0,4)
../linuxthreads/sysdeps/pthread/bits/pthreadtypes.h
../sysdeps/unix/sysv/linux/bits/sched.h
__sched_param:T(10,1)=s4__sched_priority:(0,1),0,32;;
_pthread_fastlock:T(9,1)=s8__status:(0,3),0,32;__spinlock:(0,1),32,32;;
_pthread_descr:t(9,2)=(9,3)=*(9,4)=xs_pthread_descr_struct:
__pthread_attr_s:T(9,5)=s36__detachstate:(0,1),0,32;__schedpolicy:(0,1),32,32;__sched
 param:(10,1),64,32;__inheritsched:(0,1),96,32;_
_scope:(0,1),128,32;__guardsize:(8,1),160,32;__stackaddr_set:(0,1),192,32;__stackaddr:
 (0,19),224,32;__stacksize:(8,1),256,32;;
pthread_attr_t:t(9,6)=(9,5)
pthread_cond_t:t(9,7)=(9,8)=s12__c_lock:(9,1),0,64;__c_waiting:(9,2),64,32;;
pthread_condattr_t:t(9,9)=(9,10)=s4__dummy:(0,1),0,32;;
pthread_key_t:t(9,11)=(0,4)
pthread_mutex_t:t(9,12)=(9,13)=s24__m_reserved:(0,1),0,32;__m_count:(0,1),32,32;__m
 _owner:(9,2),64,32;__m_kind:(0,1),96,32;__m_lock:
(9,1),128,64;;
pthread_mutexattr_t:t(9,14)=(9,15)=s4__mutexkind:(0,1),0,32;;
pthread_once_t:t(9,16)=(0,1)
_pthread_rwlock_t:T(9,17)=s32__rw_lock:(9,1),0,64;__rw_readers:(0,1),64,32;__rw_write
 r:(9,2),96,32;__rw_read_waiting:(9,2),128,32;__
rw_write_waiting:(9,2),160,32;__rw_kind:(0,1),192,32;__rw_pshared:(0,1),224,32;;
pthread_rwlock_t:t(9,18)=(9,17)
pthread_rwlockattr_t:t(9,19)=(9,20)=s8__lockkind:(0,1),0,32;__pshared:(0,1),32,32;;
pthread_spinlock_t:t(9,21)=(0,1)
pthread_barrier_t:t(9,22)=(9,23)=s20__ba_lock:(9,1),0,64;__ba_required:(0,1),64,32;__ba
 _present:(0,1),96,32;__ba_waiting:(9,2),128,3
2;;
pthread_barrierattr_t:t(9,24)=(9,25)=s4__pshared:(0,1),0,32;;
pthread_t:t(9,26)=(0,5)
wchar_t:t(11,1)=(0,3)
wint_t:t(11,2)=(0,4)
../include/wchar.h
../wcsmbs/wchar.h
../sysdeps/unix/sysv/linux/i386/bits/wchar.h
__mbstate_t:t(13,1)=(13,2)=s8__count:(0,1),0,32;__value:(13,3)=u4__wch:(11,2),0,32;__
 wchb:(13,4)=ar(4,33);0;3;(0,2),0,32;;,32,32;;
_G_fpos_t:t(3,1)=(3,2)=s12__pos:(4,23),0,32;__state:(13,1),32,64;;
_G_fpos64_t:t(3,3)=(3,4)=s16__pos:(4,54),0,64;__state:(13,1),64,64;;
../include/gconv.h
../iconv/gconv.h
:T(17,1)=e__GCONV_OK:0,__GCONV_NOCONV:1,__GCONV_NODB:2,__GCONV_NO
 MEM:3,__GCONV_EMPTY_INPUT:4,__GCONV_FULL_OUTPUT:5,__GCONV_ILLEGA
 L_
```

```
INPUT:6,__GCONV_INCOMPLETE_INPUT:7,__GCONV_ILLEGAL_DESCRIPTOR:8,__
 GCONV_INTERNAL_ERROR:9,;
 :T(17,2)=e__GCONV_IS_LAST:1,__GCONV_IGNORE_ERRORS:2,;
__gconv_fct:t(17,3)=(17,4)=*(17,5)=f(0,1)
__gconv_init_fct:t(17,6)=(17,7)=*(17,8)=f(0,1)
__gconv_end_fct:t(17,9)=(17,10)=*(17,11)=f(0,20)
__gconv_trans_fct:t(17,12)=(17,13)=*(17,14)=f(0,1)
__gconv_trans_context_fct:t(17,15)=(17,16)=*(17,17)=f(0,1)
__gconv_trans_query_fct:t(17,18)=(17,19)=*(17,20)=f(0,1)
__gconv_trans_init_fct:t(17,21)=(17,22)=*(17,23)=f(0,1)
__gconv_trans_end_fct:t(17,24)=(17,25)=*(17,26)=f(0,20)
__gconv_trans_data:T(17,27)=s20__trans_fct:(17,12),0,32;__trans_context_fct:(17,15),32
 ,32;__trans_end_fct:(17,24),64,32;__data:(0,19
),96,32;__next:(17,28)=*(17,27),128,32;;
__gconv_step:T(17,29)=s56__shlib_handle:(17,30)=*(17,31)=xs__gconv_loaded_object:,
 0,32;__modname:(17,32)=*(0,2),32,32;__counter:(0,1
),64,32;__from_name:(4,36),96,32;__to_name:(4,36),128,32;__fct:(17,3),160,32;__init_fct
 :(17,6),192,32;__end_fct:(17,9),224,32;__min_
needed_from:(0,1),256,32;__max_needed_from:(0,1),288,32;__min_needed_to:(0,1),320
 ,32;__max_needed_to:(0,1),352,32;__stateful:(0,1),3
84,32;__data:(0,19),416,32;;
__gconv_step_data:T(17,33)=s36__outbuf:(17,34)=*(0,11),0,32;__outbufend:(17,34),32,3
 2;__flags:(0,1),64,32;__invocation_counter:(0,1)
,96,32;__internal_use:(0,1),128,32;__statep:(17,35)=*(13,1),160,32;__state:(13,1),192,64
 ;__trans:(17,28),256,32;;
__gconv_info:T(17,36)=s8__nsteps:(8,1),0,32;__steps:(17,37)=*(17,29),32,32;__data:(17,
 38)=ar(4,33);0;-1;(17,33),64,0;;
__gconv_t:t(17,39)=(17,40)=*(17,36)
_G_iconv_t:t(3,5)=(3,6)=u44__cd:(17,36),0,64;__combined:(3,7)=s44__cd:(17,36),0,64;_
 _data:(17,33),64,288;;,0,352;;
_G_int16_t:t(3,8)=(0,8)
_G_int32_t:t(3,9)=(0,1)
_G_uint16_t:t(3,10)=(0,9)
_G_uint32_t:t(3,11)=(0,4)
_IO_stdin_used:G(0,1)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
01.01
01.01
01.01
01.01
01.01
01.01
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.hash
```

```
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rel.dyn
.rel.plt
.init
.plt
.text
.fini
.rodata
.data
.eh_frame
.ctors
.dtors
.got
.dynamic
.sbss
.bss
.stab
.stabstr
.comment
.note
ÿÿÿo
þÿÿo
initfini.c
/usr/bin/gcc2_compiled.
call_gmon_start
init.c
crtstuff.c
__DTOR_LIST__
completed.1
__do_global_dtors_aux
__EH_FRAME_BEGIN__
fini_dummy
object.2
frame_dummy
init_dummy
force_to_data
__CTOR_LIST__
__do_global_ctors_aux
__CTOR_END__
__DTOR_END__
__FRAME_END__
test.c
_DYNAMIC
__register_frame_info@@GLIBC_2.0
_fp_hw
_init
__deregister_frame_info@@GLIBC_2.0
_start
__bss_start
main
```

```
__libc_start_main@@GLIBC_2.0
data_start
printf@@GLIBC_2.0
_fini
__cxa_finalize@@GLIBC_2.1.3
_edata
_GLOBAL_OFFSET_TABLE_
_end
_IO_stdin_used
__data_start
__gmon_start__
```

The comments that were added to *test.c* did not show up in our output from *strings –a*. The reason for this is that the compiler does not include the comments when compiling the C source code into assembly language and therefore it will not be present in the final binary produced by the linker. To verify this we will create just the assembly language code for *test.c*.

```
[root@localhost tools]# /bin/cat test.c

/*This is a test
of the strings
binary parsing
utility*/

int main()
{
printf("Hello world\n");
return(0);
}

[root@localhost tools]# /usr/bin/gcc -S test.c
[root@localhost tools]# /bin/cat test.s
        .file   "test.c"
        .version        "01.01"
/usr/bin/gcc2_compiled.:
                .section        .rodata
.LC0:
        .string "Hello World\n"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        subl    $12, %esp
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        movl    $0, %eax
```

```
        leave
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)"
```

In the above assembly language code we don't see any reference to the comment that we added. Since this would be the fist step in the compilation of the binary and we know that it does not exist in the assembly language code we can conclude that */usr/bin/gcc* does not process the comments.

Next we will include some static information in the C code. We will use the *#define* preprocessor, global variables and arrays and local variables and arrays. We will give define each piece of static information differently so that we can be sure of what we are seeing. We will also define two separate pieces of static date, one for integers and one for characters.

```
[root@localhost tools]# /bin/cat test.c

/***********************************
*This is a define preprocessor*
*statement using characters.  *
***********************************/

#define SANS "This_is_a_test"

/***********************************
*This is a define preprocessor*
*statement using integers.      *
***********************************/

#define SANS2 333

/*********************************
*Define an integer varable.*
*********************************/

int integer_var;

/*****************************
*Define a character array.*
*****************************/

char char_array[] = "This is a test of the global char array";

/*****************************
*Define an integer array.*
*This array will hold 4    *
*integers.                 *
*****************************/

int int_array[] = {1, 2, 3, 4,};
```

```
int main ()
{
	{int test_of_local_variable;
	int test_of_local_array[] = {11,22,33,44};
	char test_of_local_char_array[] ="This is a test of the local char array";

	test_of_local_variable = 222;

	printf("Test of the local character array: %s\n", test_of_local_char_array);
		printf("Test of the local integer variable: %d\n", test_of_local_variable);
		printf("Test of the local integer array %d\n", test_of_local_array);
	}

integer_var = SANS2;
printf("Test of global integer variable: %d\n", integer_var);
printf("Test of the global character array: %s\n", char_array);
printf("Test of define statement: %s\n", SANS);
return(0);
}

[root@localhost tools]# /usr/bin/gcc -otest test.c
[root@localhost tools]# /usr/bin/strings -a test
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
printf
__cxa_finalize
__deregister_frame_info
_IO_stdin_used
__libc_start_main
__register_frame_info
GLIBC_2.1.3
GLIBC_2.0
ÉÃÿ5ø
éàÿÿÿÿ%
éÐÿÿÿÿ%
éÀÿÿÿÿ%
é°ÿÿÿÿ%
é ÿÿÿ
äðPTRhp
QVh`
è«ÿÿÿô
åSPè
]üÉÃ
Éuê¸
è ÿÿÿ
èËþÿÿ
}Ø¾¸
}¨¾À
ó¤ÇEôÞ
E¨Ph
ÿuôh@
```

è}þÿÿ
EØPh
èiþÿÿ
èIþÿÿ
è4þÿÿ
eø^_]Ã
øÿuôX[]ÃU
åSRè
è'þÿÿ
]üÉÃ
This is a test of the local char array
Test of the local character array: %s
Test of the local integer variable: %d
Test of the local integer array %d
Test of global integer variable: %d
Test of the global character array: %s
This_is_a_test
Test of define statement: %s
This is a test of the global char array
ÿÿÿÿ
ÿÿÿÿ
þÿÿo
ÿÿÿo
ðÿÿor
init.c
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/csu/
/usr/bin/gcc2_compiled.
int:t(0,1)=r(0,1);-2147483648;2147483647;
char:t(0,2)=r(0,2);0;127;
long int:t(0,3)=r(0,3);-2147483648;2147483647;
unsigned int:t(0,4)=r(0,4);0000000000000;0037777777777;
long unsigned int:t(0,5)=r(0,5);0000000000000;0037777777777;
long long
int:t(0,6)=@s64;r(0,6);01000000000000000000000;07777777777777777777;
long long unsigned int:t(0,7)=@s64;r(0,7);0000000000000;01777777777777777777777;
short int:t(0,8)=@s16;r(0,8);-32768;32767;
short unsigned int:t(0,9)=@s16;r(0,9);0;65535;
signed char:t(0,10)=@s8;r(0,10);-128;127;
unsigned char:t(0,11)=@s8;r(0,11);0;255;
float:t(0,12)=r(0,1);4;0;
double:t(0,13)=r(0,1);8;0;
long double:t(0,14)=r(0,1);12;0;
complex int:t(0,15)=s8real:(0,1),0,32;imag:(0,1),32,32;;
complex float:t(0,16)=r(0,16);8;0;
complex double:t(0,17)=r(0,17);16;0;
complex long double:t(0,18)=r(0,18);24;0;
__builtin_va_list:t(0,19)=*(0,20)=(0,20)
../include/libc-symbols.h
/usr/src/build/40453-i386/BUILD/glibc-2.2.4/build-i386-linux/config.h
../sysdeps/gnu/_G_config.h
../sysdeps/unix/sysv/linux/bits/types.h
../include/features.h
../include/sys/cdefs.h

```
../misc/sys/cdefs.h
/usr/lib//usr/bin/gcc-lib/i386-redhat-linux/2.96/include/stddef.h
size_t:t(8,1)=(0,4)
__u_char:t(4,1)=(0,11)
__u_short:t(4,2)=(0,9)
__u_int:t(4,3)=(0,4)
__u_long:t(4,4)=(0,5)
__u_quad_t:t(4,5)=(0,7)
__quad_t:t(4,6)=(0,6)
__int8_t:t(4,7)=(0,10)
__uint8_t:t(4,8)=(0,11)
__int16_t:t(4,9)=(0,8)
__uint16_t:t(4,10)=(0,9)
__int32_t:t(4,11)=(0,1)
__uint32_t:t(4,12)=(0,4)
__int64_t:t(4,13)=(0,6)
__uint64_t:t(4,14)=(0,7)
__qaddr_t:t(4,15)=(4,16)=*(4,6)
__dev_t:t(4,17)=(4,5)
__uid_t:t(4,18)=(4,3)
__gid_t:t(4,19)=(4,3)
__ino_t:t(4,20)=(4,4)
__mode_t:t(4,21)=(4,3)
__nlink_t:t(4,22)=(4,3)
__off_t:t(4,23)=(0,3)
__loff_t:t(4,24)=(4,6)
__pid_t:t(4,25)=(0,1)
__ssize_t:t(4,26)=(0,1)
__rlim_t:t(4,27)=(4,4)
__rlim64_t:t(4,28)=(4,5)
__id_t:t(4,29)=(4,3)
__fsid_t:t(4,30)=(4,31)=s8__val:(4,32)=ar(4,33)=r(4,33);0000000000000;0037777777777
;;0;1;(0,1),0,64;;
__daddr_t:t(4,34)=(0,1)
__caddr_t:t(4,35)=(4,36)=*(0,2)
__time_t:t(4,37)=(0,3)
__useconds_t:t(4,38)=(0,4)
__suseconds_t:t(4,39)=(0,3)
__swblk_t:t(4,40)=(0,3)
__clock_t:t(4,41)=(0,3)
__clockid_t:t(4,42)=(0,1)
__timer_t:t(4,43)=(0,1)
__key_t:t(4,44)=(0,1)
__ipc_pid_t:t(4,45)=(0,9)
__blksize_t:t(4,46)=(0,3)
__blkcnt_t:t(4,47)=(0,3)
__blkcnt64_t:t(4,48)=(4,6)
__fsblkcnt_t:t(4,49)=(4,4)
__fsblkcnt64_t:t(4,50)=(4,5)
__fsfilcnt_t:t(4,51)=(4,4)
__fsfilcnt64_t:t(4,52)=(4,5)
__ino64_t:t(4,53)=(4,5)
__off64_t:t(4,54)=(4,24)
```

```
__t_scalar_t:t(4,55)=(0,3)
__t_uscalar_t:t(4,56)=(0,5)
__intptr_t:t(4,57)=(0,1)
__socklen_t:t(4,58)=(0,4)
../linuxthreads/sysdeps/pthread/bits/pthreadtypes.h
../sysdeps/unix/sysv/linux/bits/sched.h
__sched_param:T(10,1)=s4__sched_priority:(0,1),0,32;;
_pthread_fastlock:T(9,1)=s8__status:(0,3),0,32;__spinlock:(0,1),32,32;;
_pthread_descr:t(9,2)=(9,3)=*(9,4)=xs_pthread_descr_struct:
__pthread_attr_s:T(9,5)=s36__detachstate:(0,1),0,32;__schedpolicy:(0,1),32,32;__sched
param:(10,1),64,32;__inheritsched:(0,1),96,32;_
_scope:(0,1),128,32;__guardsize:(8,1),160,32;__stackaddr_set:(0,1),192,32;__stackaddr
:(0,19),224,32;__stacksize:(8,1),256,32;;
pthread_attr_t:t(9,6)=(9,5)
pthread_cond_t:t(9,7)=(9,8)=s12__c_lock:(9,1),0,64;__c_waiting:(9,2),64,32;;
pthread_condattr_t:t(9,9)=(9,10)=s4__dummy:(0,1),0,32;;
pthread_key_t:t(9,11)=(0,4)
pthread_mutex_t:t(9,12)=(9,13)=s24__m_reserved:(0,1),0,32;__m_count:(0,1),32,32;__
m_owner:(9,2),64,32;__m_kind:(0,1),96,32;__m_lock:
(9,1),128,64;;
pthread_mutexattr_t:t(9,14)=(9,15)=s4__mutexkind:(0,1),0,32;;
pthread_once_t:t(9,16)=(0,1)
_pthread_rwlock_t:T(9,17)=s32__rw_lock:(9,1),0,64;__rw_readers:(0,1),64,32;__rw_writ
er:(9,2),96,32;__rw_read_waiting:(9,2),128,32;__
rw_write_waiting:(9,2),160,32;__rw_kind:(0,1),192,32;__rw_pshared:(0,1),224,32;;
pthread_rwlock_t:t(9,18)=(9,17)
pthread_rwlockattr_t:t(9,19)=(9,20)=s8__lockkind:(0,1),0,32;__pshared:(0,1),32,32;;
pthread_spinlock_t:t(9,21)=(0,1)
pthread_barrier_t:t(9,22)=(9,23)=s20__ba_lock:(9,1),0,64;__ba_required:(0,1),64,32;__b
a_present:(0,1),96,32;__ba_waiting:(9,2),128,3
2;;
pthread_barrierattr_t:t(9,24)=(9,25)=s4__pshared:(0,1),0,32;;
pthread_t:t(9,26)=(0,5)
wchar_t:t(11,1)=(0,3)
wint_t:t(11,2)=(0,4)
../include/wchar.h
../wcsmbs/wchar.h
../sysdeps/unix/sysv/linux/i386/bits/wchar.h
__mbstate_t:t(13,1)=(13,2)=s8__count:(0,1),0,32;__value:(13,3)=u4__wch:(11,2),0,32;__
wchb:(13,4)=ar(4,33);0;3;(0,2),0,32;;,32,32;;
_G_fpos_t:t(3,1)=(3,2)=s12__pos:(4,23),0,32;__state:(13,1),32,64;;
_G_fpos64_t:t(3,3)=(3,4)=s16__pos:(4,54),0,64;__state:(13,1),64,64;;
../include/gconv.h
../iconv/gconv.h

:T(17,1)=e__GCONV_OK:0,__GCONV_NOCONV:1,__GCONV_NODB:2,__GCONV_NO
MEM:3,__GCONV_EMPTY_INPUT:4,__GCONV_FULL_OUTPUT:5,__GCONV_ILLEGA
L_
INPUT:6,__GCONV_INCOMPLETE_INPUT:7,__GCONV_ILLEGAL_DESCRIPTOR:8,__
GCONV_INTERNAL_ERROR:9,;
 :T(17,2)=e__GCONV_IS_LAST:1,__GCONV_IGNORE_ERRORS:2,;
__gconv_fct:t(17,3)=(17,4)=*(17,5)=f(0,1)
__gconv_init_fct:t(17,6)=(17,7)=*(17,8)=f(0,1)
```

__gconv_end_fct:t(17,9)=(17,10)=*(17,11)=f(0,20)
__gconv_trans_fct:t(17,12)=(17,13)=*(17,14)=f(0,1)
__gconv_trans_context_fct:t(17,15)=(17,16)=*(17,17)=f(0,1)
__gconv_trans_query_fct:t(17,18)=(17,19)=*(17,20)=f(0,1)
__gconv_trans_init_fct:t(17,21)=(17,22)=*(17,23)=f(0,1)
__gconv_trans_end_fct:t(17,24)=(17,25)=*(17,26)=f(0,20)
__gconv_trans_data:T(17,27)=s20__trans_fct:(17,12),0,32;__trans_context_fct:(17,15),32,32;__trans_end_fct:(17,24),64,32;__data:(0,19
),96,32;__next:(17,28)=*(17,27),128,32;;
__gconv_step:T(17,29)=s56__shlib_handle:(17,30)=*(17,31)=xs__gconv_loaded_object:,0,32;__modname:(17,32)=*(0,2),32,32;__counter:(0,1
),64,32;__from_name:(4,36),96,32;__to_name:(4,36),128,32;__fct:(17,3),160,32;__init_fct:(17,6),192,32;__end_fct:(17,9),224,32;__min_
needed_from:(0,1),256,32;__max_needed_from:(0,1),288,32;__min_needed_to:(0,1),320,32;__max_needed_to:(0,1),352,32;__stateful:(0,1),3
84,32;__data:(0,19),416,32;;
__gconv_step_data:T(17,33)=s36__outbuf:(17,34)=*(0,11),0,32;__outbufend:(17,34),32,32;__flags:(0,1),64,32;__invocation_counter:(0,1)
,96,32;__internal_use:(0,1),128,32;__statep:(17,35)=*(13,1),160,32;__state:(13,1),192,64;__trans:(17,28),256,32;;
__gconv_info:T(17,36)=s8__nsteps:(8,1),0,32;__steps:(17,37)=*(17,29),32,32;__data:(17,38)=ar(4,33);0;-1;(17,33),64,0;;
__gconv_t:t(17,39)=(17,40)=*(17,36)
_G_iconv_t:t(3,5)=(3,6)=u44__cd:(17,36),0,64;__combined:(3,7)=s44__cd:(17,36),0,64;__data:(17,33),64,288;;,0,352;;
_G_int16_t:t(3,8)=(0,8)
_G_int32_t:t(3,9)=(0,1)
_G_uint16_t:t(3,10)=(0,9)
_G_uint32_t:t(3,11)=(0,4)
_IO_stdin_used:G(0,1)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
01.01
01.01
01.01
01.01
01.01
01.01
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rel.dyn

```
.rel.plt
.init
.plt
.text
.fini
.rodata
.data
.eh_frame
.ctors
.dtors
.got
.dynamic
.sbss
.bss
.stab
.stabstr
.comment
.note
ÿÿÿo
þÿÿo
initfini.c
/usr/bin/gcc2_compiled.
call_gmon_start
init.c
crtstuff.c
__DTOR_LIST__
completed.1
__do_global_dtors_aux
__EH_FRAME_BEGIN__
fini_dummy
object.2
frame_dummy
init_dummy
force_to_data
__CTOR_LIST__
__do_global_ctors_aux
__CTOR_END__
__DTOR_END__
__FRAME_END__
test.c
_DYNAMIC
__register_frame_info@@GLIBC_2.0
_fp_hw
integer_var
_init
__deregister_frame_info@@GLIBC_2.0
_start
char_array
int_array
__bss_start
main
__libc_start_main@@GLIBC_2.0
data_start
```

```
printf@@GLIBC_2.0
_fini
__cxa_finalize@@GLIBC_2.1.3
_edata
_GLOBAL_OFFSET_TABLE_
_end
_IO_stdin_used
__data_start
__gmon_start__
```

From the above output we can see all the character information contained in the arrays, global and local, and the first parameter passed to the *printf* function. The reason that we will always see the first parameter passed to the *printf* function is that the data is read-only. Functions in the C programming language are not able to modify any data that is passed to it, unless the data is passed via pointers. This is the reason that the first argument to the *printf* function is read-only. To verify this we will look at the output of *objdump*. We will run *objdump* so that it only looks at the read-only portion of the binary. In the ELF format this would be the *.rodata* section.

```
[root@localhost tools]# /bin/cat test.c
#include <stdio.h>
/**********************************
*This is a define preprocessor*
*statement using characters.  *
**********************************/

#define SANS "This_is_a_test"

/**********************************
*This is a define preprocessor*
*statement using integers.      *
**********************************/

#define SANS2 333

/*******************************
*Define an integer varable.*
*******************************/

int integer_var;

/*******************************
*Define a character array.*
*******************************/

char char_array[] = "This is a test of the global char array";

/*****************************
*Define an integer array.*
*This array will hold 100 *
```

```
*integers.                    *
**************************/

int int_array[] = {1, 2, 3, 4,};

int main ()
{
        {int test_of_local_variable;
        int test_of_local_array[] = {11,22,33,44};
        char test_of_local_char_array[] ="This is a test of the local char array";

        test_of_local_variable = 222;

        printf("Test of the local character array: %s\n", test_of_local_char_array);
                printf("Test of the local integer variable: %d\n", test_of_local_variable);
                printf("Test of the local integer array %d\n", test_of_local_array);
        }

integer_var = SANS2;
printf("Test of global integer variable: %d\n", integer_var);
printf("Test of the global character array: %s\n", char_array);
printf("Test of define statement: %s\n", SANS);
return(0);
}

[root@localhost tools]# /usr/bin/objdump -j .rodata -s test

test:     file format elf32-i386

Contents of section .rodata:
 80485a0 03000000 01000200 00000000 00000000  ................
 80485b0 00000000 00000000 00000000 00000000  ................
 80485c0 54686973 20697320 61207465 7374206f  This is a test o
 80485d0 66207468 65206c6f 63616c20 63686172  f the local char
 80485e0 20617272 61790000 00000000 00000000   array.........
 80485f0 00000000 00000000 00000000 00000000  ................
 8048600 54657374 206f6620 74686520 6c6f6361  Test of the loca
 8048610 6c206368 61726163 74657220 61727261  l character arra
 8048620 793a2025 730a0000 00000000 00000000  y: %s..........
 8048630 00000000 00000000 00000000 00000000  ................
 8048640 54657374 206f6620 74686520 6c6f6361  Test of the loca
 8048650 6c20696e 74656765 72207661 72696162  l integer variab
 8048660 6c653a20 25640a00 00000000 00000000  le: %d..........
 8048670 00000000 00000000 00000000 00000000  ................
 8048680 54657374 206f6620 74686520 6c6f6361  Test of the loca
 8048690 6c20696e 74656765 72206172 72617920  l integer array
 80486a0 25640a00 00000000 00000000 00000000  %d..............
 80486b0 00000000 00000000 00000000 00000000  ................
 80486c0 54657374 206f6620 676c6f62 616c2069  Test of global i
 80486d0 6e746567 65722076 61726961 626c653a  nteger variable:
 80486e0 2025640a 00000000 00000000 00000000   %d.............
 80486f0 00000000 00000000 00000000 00000000  ................
 8048700 54657374 206f6620 74686520 676c6f62  Test of the glob
```

```
 8048710 616c2063 68617261 63746572 20617272  al character arr
 8048720 61793a20 25730a00 54686973 5f69735f  ay: %s..This_is_
 8048730 615f7465 73740054 65737420 6f662064  a_test.Test of d
 8048740 6566696e 65207374 6174656d 656e743a  efine statement:
 8048750 2025730a 00000000 00000000 00000000   %s.............
[root@localhost tools]#
```

We can't see any of the integer information, regardless of variable or array. The information is in the binary, but is not kept in a form that can't be read by *strings*. To find the information the analyst would need to know more about how *gcc* compiles the C source code and assembly, both of which are outside the scope of this practical.

From the information that we saw from our testing we can conclude that:

1. We will see any strings or character arrays that are in the source code.
2. We can see the name of any global variables.
3. We can see what version of the C library was used.
4. We can see what version of /USR/BIN/GCC was used.
5. We can see what was used to link the binary.
6. We can see the original C source code file.

The *strings* utility gave us all the information that we expected. The results were very good and the analysis provided us with good information about the *strings* utility.

Analysis

The information that is gathered by *strings* is ASCII text it is very easy to interpret and will be helpful in determining what type of system the binary was compiled on and what was used to compile it. This will be helpful if the analyst is going to reverse engineer the binary. The forensic analyst might also gain some insight into how the binary is suppose to be run (i.e. options, switches), without to actually have to run it.

While the information that is gathered by *strings* can be very helpful, none of the information should be taken as fact. A non-skilled hacker could easily add a *printf* statement with bogus information to throw off the forensic analyst. A skilled programmer could compile his/her C source code into assembly and edit the assembly code so that the information contained in the *.comment* section is incorrect. From my experience most hackers only *strip* their binaries to make the reverse engineering harder. The Teso (http://www.team-teso.net/) security group has created an ELF encryption API, called burneye, that makes it easier for programmers to obfuscate their ELF binaries. There are programs that can strip the burneye portion off a binary to reveal the actual binary.

The forensic analysis should look at all the output from *strings* and see if there is anything of interest and take note of it. After the analyst has all the

information that seems valuable he/she can start to investigate each piece of information and see where it leads. When using *strings* on the unknown binary, *sn.dat*, in the second part of the practical the information gained helped in finding out what the binary was without any reverse engineering. This is not all ways the case though.

Presentation

The *strings* utility does not have any way for the output to be saved. The forensic analyst has a couple of choices though. He/she could either log the output through a terminal emulator like SecureCRT or redirect the output to a file. Either way all of the output will be saved to an ASCII text file. When I use the *strings* utility I look at each line and determine the usefulness of it. I then copy the valuable information to a separate ASCII text file and then place a comment prior to each line. I also separate each line with a space so that we can see what the comments are referring to. Below is an example of the formatting that I would use:

```
/*Unkown information. Possible character array.*/
eø^_]ÃThis is a test of the global char array
01.01
This is a test of the local char array

/*The first part of the printf statement.*/
Test of the local character array: %s
Test of the local integer variable: %d
Test of the local integer array %d
Test of global integer variable: %d
Test of the global character array: %s

/*Unkown information. Possible character array.*/
This_is_a_test

/*The first part of the printf statement.*/
Test of define statement: %s

/*Part of the .comment section. Run objdump to verify.*/
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)

/*ELF section headers.*/
.symtab
.strtab
.shstrtab
.text
.rel.text
.data
.bss
.note
.rodata
.comment
```

```
/*The original C source code file that the binary was compiled from.*/
test.c

/*Possible names of global information.*/
char_array
int_array
main

/*Extra evidence that the printf function was used.*/
printf

/*Possible names of global information.*/
integer_var
```

Conclusion

From the test that was conducted we can see that *strings* can assist us greatly in a forensic investigation. It accurately found any character array or string information and the first argument that was passed to the *printf* function. It also assisted us in finding out what libraries and compilers were used. If the binary is not stripped then the original C source code file name will be visible.

Depending on if this file was to be part of an incident response tool kit or if it were part of forensic workstation would dictate how *strings* would have to be compiled and how *strings* would affect the files that are searched.

If *strings* were to be part of an incident response tool kit it would have to be statically compiled. If it were not then it might use files from the victim system. This means that the attacker could place trojaned versions of the shared libraries on the victim system and alter the usefulness of *strings* and any other utility that uses those shared libraries. It would also change the atime of the files that were looked at. Depending on the situation this could severely affect the outcome of the investigation.

If *strings* were to be part of a forensic workstation then it would not have to be statically compiled because the evidence would be some type of device that is not normally part of the workstation (i.e. *dd* image, secondary hard drive). The shared libraries would be part of a known good system and would not adversely affect the outcome of the investigation.

It would be recommended that *strings* be statically compiled and stripped. Once this is done a md5 checksum should be made of *strings*. This could be said of all the tools/utilities used in the forensic investigation. The checksums would be used prior to using any tools to ensure that they have not be modified in any way. It would also be beneficial for some agency or company to certify certain tools/utilities and provide the certified versions to the community along with md5 checksums. This would ensure that the tool/utility used has been certified and that it did it's function properly.

Additional Information

Strings is part of the *binutils* package (http://sources.redhat.com/binutils/). The *binutils* package is very useful in the reverse engineering process. For a statically linked version of *strings* check out http://www.incident-response.org/irtoolkits.htm.

Part II

Preparation

Prior to analyzing the binary I made sure that all the components are in place to do the analysis. The analysis workstation is configured as follows:

1. Windows 2000 SP1: This workstation has ActiveState's PERL 5.6.0, UltraEdite32 7.10a, 3Cdaemon v2/rev10 (FTP server), SecureCRT 3.3.2, VMWare 3.1.1 build 1790 and Cygwin 1.3.10 installed on it.
2. Redhat 7.2 (VMWare, host-only mode): This is a default install of Redhat 7.2 from the CD. Everything was installed. *Mac_daddy.pl* was also installed on the system.

I created a directory (Windows => C:\Docuements\Forensics\TOOLS\, Linux => /usr/tools/binary) on both workstations for the analysis. I downloaded the file (sn.zip) from the GIAC web site. There was no MD5 hash of the file on the web site to verify the integrity of the download so I had to assume that everything was downloaded correctly. I ran md5sum to get an MD5 hash prior to FTPing the file to the Linux station. (NOTE: For part two of the practical all output from SecureCRT or the Windows command line will be shown in a 10 point Arial to eliminate the line wrap and improve readability.)

```
D:\Windows and Advanced Forensics\response_kit\win2k_xp>md5sum.exe
c:\Documents\forensics\TOOLS\sn.zip
\5fea57f2a1546bc391c6b9cb1bbfc452 *c:\\Documents\\forensics\\TOOLS\\sn.zip
```

I FTP'ed the file from the Windows station to the Linux station and ran *md5sum* again to verify the integrity of the file

```
[root@localhost linux_x86_static]# ./md5sum      /usr/tools/binary/sn.zip
5fea57f2a1546bc391c6b9cb1bbfc452  /usr/tools/sn.zip
```

Once I verified the integrity of the file I created a sterilized floppy from the Linux host and formatted it as e2fs.

```
[root@localhost tools]# dd if=/dev/zero of=/dev/fd0
```

```
[root@localhost tools]# mke2fs /dev/fd0
```

I then mounted the floppy and unzipped the file *sn.zip* to the floppy and then unmounted the floppy.

```
[root@localhost tools]# mount /mnt/floppy
[root@localhost tools]# unzip –d /mnt/floppy       /usr/tools/binary/sn.zip
[root@localhost tools]# umount /mnt/floppy
```

I made the floppy read only via the tab on the floppy disk and then remounted it as I would have any partition that required a forensics investigation.

```
[root@localhost tools]# mount -o ro,nodev,noexec,noatime        /mnt/floppy
```

I also unzipped a copy of the file *sn.zip* into the */usr/tools/binary* directory for the disassembly portion of the forensics examination.

Binary Details

The name of the original file is *sn*. The file that was part of the *sn.zip* file is named *sn.dat.* The output from */usr/bin/md5sum* (sn.md5)shows that the file name at the time that */usr/bin/md5sum* was run on the file *sn*.

```
[root@localhost binary]# /bin/cat sn.md5
0e954f43fd73f56e812a7285f32e41d3  sn
```

Running *mac_daddy.pl* shows that the last modified and access times on *sn.dat* are both April 11th, 2002 at 9:29:58. The last change time for *sn.dat* was September 16th, 2002 at 07:10:03, which is the time that *sn.zip* was unzipped. (NOTE: This assumes that the system clock on the host that the binary was found on was correct.)

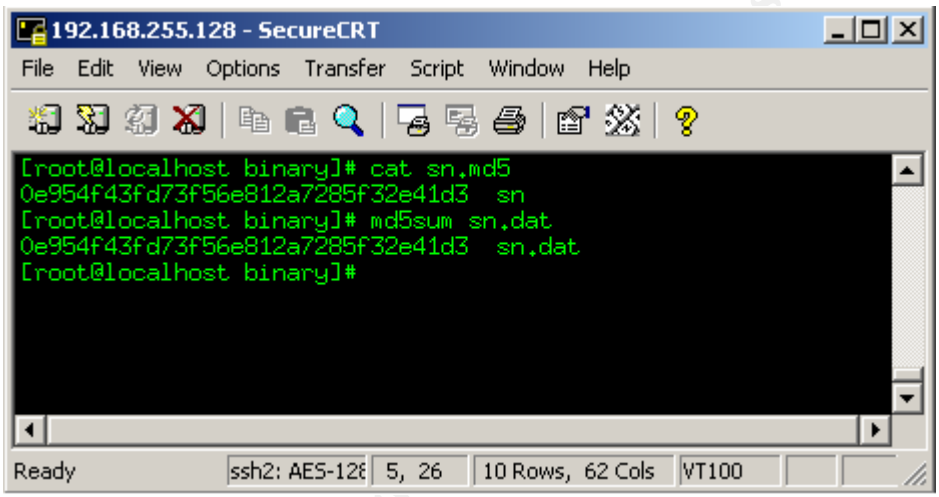
```
[root@localhost mac_daddy]# ./mac_daddy.pl /mnt/floppy | grep sn
Apr 11 2002 09:29:52          37 ma.  -rw-rw-rw- root     root     /mnt/floppy/sn.md5
Apr 11 2002 09:29:58    399124 ma.  -rw-rw-rw- root     root     /mnt/floppy/sn.dat
Sep 16 2002 07:10:03    399124   ..c  -rw-rw-rw- root     root     /mnt/floppy/sn.dat
                                37   ..c   -rw-rw-rw- root     root     /mnt/floppy/sn.md5
```

The owner of the file is *root*. This file was most likely not on the system when the attacker compromised it. The attacker transferred this file from some other system under their control. The file would show what account the file was created under. The only time that the UID/GID would be shown and not the name of the UID/GID is if the system that the file was mounted on did not have a corresponding UID/GID.

The file is 399,124 bytes. This can be seen in the output from *ls –la* and - *mac_daddy.pl*:

```
[root@localhost mac_daddy]# ls -la /usr/tools/binary/sn.dat
-rwxr-xr-x    1 root     root       399124 Apr 11 09:29 /usr/tools/binary/sn.dat
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep sn.dat
Apr 11 2002 09:29:58   399124 m.. -rwxr-xr-x root     root     /usr/tools/binary/sn.dat
Sep 16 2002 18:02:39   399124 ..c -rwxr-xr-x root     root     /usr/tools/binary/sn.dat
Sep 16 2002 18:07:33   399124 .a. -rwxr-xr-x root     root     /usr/tools/binary/sn.dat
```

Even though we are showing the output from the SecureCRT session we will provide a screenshot (bitmap image) for the md5 checksum output.



To see what keywords are associated with the file we ran */usr/bin/strings –a* on *sn.dat* and came up with the following text that helped in determining what the binary was.

```
SIOCGSTAMP: %s
bind: %s: %s
SIOCGIFHWADDR: %s
SIOCGIFMTU: %s
SIOCGIFFLAGS: %s
linux socket: %s
linux SIOCSIFFLAGS: %s
unknown physical layer type 0x%x
ld.so-1.7.0
glibc-ld.so.cache1.1
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
--=[ %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)
```

Program Description

By executing */usr/bin/file* on *sn.dat* we know that it an executable.

```
[root@localhost tools]# /usr/bin/file /mnt/floppy/sn.dat
/mnt/floppy/sn.dat: ELF 32-bit LSB executable, Intel 80386, version 1, statically linked,
stripped
```

This also tells us that the executable was statically linked. This means that the file will not use any shared libraries. The file was also stripped of all symbols. The file *sn.dat* is really a file called *sn.* This can be seen in the file *sn.md5,* which is the output from */usr/bin/md5sum.*

```
[root@localhost binary]# /bin/cat sn.md5
0e954f43fd73f56e812a7285f32e41d3  sn
```

By looking at the output from *mac_daddy.pl* we can see the MACTimes of *sn.dat*

```
[root@localhost mac_daddy]# ./mac_daddy.pl /mnt/floppy | grep sn
Apr 11 2002 09:29:52        37 ma.  -rw-rw-rw- root     root     /mnt/floppy/sn.md5
Apr 11 2002 09:29:58    399124 ma.  -rw-rw-rw- root     root     /mnt/floppy/sn.dat
Sep 16 2002 07:10:03    399124   ..c  -rw-rw-rw- root     root     /mnt/floppy/sn.dat
                            37   ..c  -rw-rw-rw- root     root     /mnt/floppy/sn.md5
```

From the output from *mac_daddy.pl* we see two pieces of information that are not consistent with an executable binary. The first is that under most normal circumstances an executable would not have the same modified and access times. The access time should be newer. When compiling a binary, all of the times (modified, access and change) will be the same on the final executable. This is because the file is created, similar to a file's MACTimes when a file is

created with */bin/touch*. If the file were to be executed then the access time would have changed, but this is not what we see in *sn.dat*. The other anomaly that is seen are the permissions of *sn.dat*. When I found out what the binary was and downloaded a copy of it and then complied it, the permissions on the binary were set to 755. The permissions on *sn.dat* are set to 666. The permissions that are set are usually associated with a normal file. There is no way that the *umask* could been set to change the permissions from 755 to 666 as the *umask* can only take away permissions, not add them. The change in permissions would have to have been done with */bin/chmod*. We will take a look at these discrepancies along with other information that we learn from *sn.dat* to reconstruct a plausible series of events. From examining the evidence we can conclude:

1. The file *sn* had the */usr/bin/md5sum* utility ran against it. This can be seen from the file *sn.md5.*
2. The file *sn.dat* was zipped (compressed). This can be seen from the file extension of the compressed file, *sn.zip.*
3. The file name *sn* was changed to *sn.dat*. This can be seen when comparing the file that had */usr/bin/md5sum* against it and the file that was part of *sn.zip*.
4. The modified time and access time of *sn.dat* are the same. This can be seen from the output of *mac_daddy.pl.*
5. The change time of *sn.dat* is the time that the *sn.zip* file was unzipped. This can be seen from the output of *mac_daddy.pl.*
6. The modified time and access time of *sn.md5* is six seconds prior to the modified time and access time of the *sn.dat* file. This can be seen from the output of *mac_daddy.pl.*
7. The change time of *sn.md5* is the time that the *sn.zip* file was unzipped. This can be seen from the output of *mac_daddy.pl.*
8. The permissions on *sn.dat* or *sn* were changed. This can be seen with either *macdaddy.pl* or */bin/ls –la*.

Based on the information above we can reconstruct a partial event timeline.

md5sum sn >>sn.md5
Rename (*cp* or *mv*) sn to sn.dat
chmod 666 sn or sn.dat
zip –r sn sn.md5 sn.dat

The reason that we don't list the file *sn* as being executed (i.e. *./sn eth0*) is because the evidence was tampered with which caused all the information regarding MACTimes to be altered. From the MACtimes on *sn.md5* we can see that */usr/bin/md5sum* was ran on *sn* before the file name was changed from *sn* to *sn.dat*, and we know that the access time on *sn* was changed during this

process. We can also conclude that the file name was changed before the file was zipped. We can't conclude at what point that */bin/chmod* was used to change the permissions on the binary.

It is evident that an unusually situation has occurred. From the current information that I have gathered I have come up with two conclusions. The first is that any of the actions carried out by an attacker would have been lost due to the way that the information was gathered. There is not enough evidence provided to prove that the system was hacked and that anything malicious was done. The second is that the evidence was not gathered in a way that is forensically correct and the evidence was tampered with prior to the collection. This leads me to believe that the file was only part of the certification and not a binary that was actually found on an actual system or honeypot. The unusual way that the binary was acquired and treated is typical certification that tests a person's level of knowledge by creating a non-real world situation.

There are five different actions that could have changed the MACtimes on the binary, running the binary, renaming (UNIX *mv* or *cp* command) the binary, calculating the md5 checksum of the binary and zipping/unzipping the binary. We will run some tests on a copy of *sn.dat* that is not on the floppy drive to derive what actions were taken on the binary.

The first binary that we will run on *sn.dat* is */bin/chmod*. The reason that we are going to run this executable first is because the current permissions of *sn.dat* will not allow us to execute it. We will be changing the permissions from 666 to 755. Before we do this though we will get the MACTimes of *sn.dat*. After running */bin/chmod* we will again get the MACTimes and see what has changed.

```
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/
Apr 11 2002 09:29:52        37 ma. -rw-rw-rw-  root   root       /usr/tools/binary/sn.md5
Apr 11 2002 09:29:58    399124 ma. -rw-rw-rw-  root   root    /usr/tools/binary/sn.dat
Sep 15 2002 15:03:50    175185 m.c -rw-r--r--  root   root    /usr/tools/binary/sn.zip
Sep 16 2002 17:59:51    399124  ..c -rw-rw-rw-  root   root    /usr/tools/binary/sn.dat
                             37 ..c -rw-rw-rw-  root   root    /usr/tools/binary/sn.md5
                         175185 .a. -rw-r--r--  root   root    /usr/tools/binary/sn.zip

[root@localhost mac_daddy]# /bin/chmod 755 /usr/tools/binary/sn.dat

[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/
Apr 11 2002 09:29:52        37 ma.  -rw-rw-rw-  root   root    /usr/tools/binary/sn.md5
Apr 11 2002 09:29:58    399124 ma.  -rwxr-xr-x  root   root    /usr/tools/binary/sn.dat
Sep 15 2002 15:03:50    175185 m.c  -rw-r--r--  root   root    /usr/tools/binary/sn.zip
Sep 16 2002 17:59:51        37  ..c -rw-rw-rw-  root   root    /usr/tools/binary/sn.md5
                         175185 .a.  -rw-r--r--  root   root    /usr/tools/binary/sn.zip
Sep 16 2002 18:02:39    399124  ..c -rwxr-xr-x  root   root    /usr/tools/binary/sn.dat
```

From the above output we can see that */bin/chmod* changes the change time of the binary.

Now we will look at is how does executing a binary change the MACtimes. We will get the MACtimes of */usr/tools/sn.dat* before and after we execute it (Note: The permissions on *sn.dat* needed to be changed before executing it.):

```
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/
[root@localhost mac_daddy]# Apr 11 2002 09:29:52       37 ma. -rw-rw-rw- root     root
/usr/tools/binary/sn.md5
Apr 11 2002 09:29:58    399124 m.. -rwxr-xr-x  root     root     /usr/tools/binary/sn.dat
Sep 15 2002 15:03:50    175185 m.c -rw-r--r--   root     root     /usr/tools/binary/sn.zip
Sep 16 2002 17:59:51        37 ..c  -rw-rw-rw- root     root     /usr/tools/binary/sn.md5
                        175185 .a. -rw-r--r--   root     root     /usr/tools/binary/sn.zip
Sep 16 2002 18:02:39    399124 ..c  -rwxr-xr-x  root     root     /usr/tools/binary/sn.dat
Sep 16 2002 18:07:07    399124 .a.  -rwxr-xr-x  root     root     /usr/tools/binary/sn.dat

[root@localhost mac_daddy]# /usr/tools/binary/sn.dat eth0
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me

[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/
Apr 11 2002 09:29:52        37 ma. -rw-rw-rw- root     root     /usr/tools/binary/sn.md5
Apr 11 2002 09:29:58    399124 m.. -rwxr-xr-x  root     root     /usr/tools/binary/sn.dat
Sep 15 2002 15:03:50    175185 m.c -rw-r--r--  root     root     /usr/tools/binary/sn.zip
Sep 16 2002 17:59:51        37 ..c -rw-rw-rw- root     root     /usr/tools/binary/sn.md5
                        175185 .a. -rw-r--r--  root     root     /usr/tools/binary/sn.zip
Sep 16 2002 18:02:39    399124 ..c -rwxr-xr-x root     root     /usr/tools/binary/sn.dat
Sep 16 2002 18:07:33    399124 .a. -rwxr-xr-x  root     root     /usr/tools/binary/sn.dat
```

We see that only the access time has changed when the binary is executed.

Next we will create two separate files and get there MACtimes. Then we will *mv* one of the files and *cp* the other file and then record the MACtimes and see what has changed.

```
[root@localhost mac_daddy]# touch /usr/tools/binary/test_file.mv
[root@localhost mac_daddy]# touch /usr/tools/binary/test_file.cp
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 18:20:57        0 mac -rw-r--r-- root     root     /usr/tools/binary/test_file.mv
Sep 16 2002 18:21:00        0 mac -rw-r--r-- root     root     /usr/tools/binary/test_file.cp

[root@localhost mac_daddy]# mv /usr/tools/binary/test_file.mv
/usr/tools/binary/test_file.mv.new
root@localhost mac_daddy]# cp /usr/tools/binary/test_file.cp
/usr/tools/binary/test_file.cp.new

[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 18:20:57        0 ma. -rw-r--r--  root     root   /usr/tools/binary/test_file.mv.new
Sep 16 2002 18:21:00        0 m.c  -rw-r--r-- root     root    /usr/tools/binary/test_file.cp
Sep 16 2002 18:28:13        0 ..c    -rw-r--r-- root     root  /usr/tools/binary/test_file.mv.new
Sep 16 2002 18:28:29        0 .a   . -rw-r--r-- root     root  /usr/tools/binary/test_file.cp
                            0 mac -rw-r--r-- root     root    /usr/tools/binary/test_file.cp.new
```

With the *mv* command we see the only thing that has changed is the change time. With the *cp* command the access time of the original file is changed while the new file gets all new MACTimes, similar to the MACTimes from *touch*.

Next we will zip/unzip a file to see what changes.

```
[root@localhost binary]# echo thisisatestoftheziputility >>testzip
[root@localhost binary]# cd ../mac_daddy/
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 19:05:48       27 mac -rw-r--r-- root     root     /usr/tools/binary/testzip
[root@localhost mac_daddy]# cd /usr/tools/binary/
[root@localhost binary]# zip -r testzip testzip
 adding: testzip (stored 0%)
[root@localhost binary]# rm -f testzip
[root@localhost binary]# unzip testzip.zip
Archive:  testzip.zip
extracting: testzip
[root@localhost binary]# cd ../mac_daddy/
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 19:05:48       27 ma. -rw-r--r-- root     root     /usr/tools/binary/testzip
Sep 16 2002 19:06:40      173 m.c -rw-r--r-- root     root     /usr/tools/binary/testzip.zip
Sep 16 2002 19:07:01       27  ..c -rw-r--r-- root     root     /usr/tools/binary/testzip
                          173 .a.  -rw-r--r-- root     root     /usr/tools/binary/testzip.zip
```

We see that only the change time changes during the zip/unzip process. The last thing that we will do is create a file and get the md5 checksum of the file and see what changes the */usr/bin/md5sum* utility makes on the file.

```
[root@localhost mac_daddy]# echo thisisatestofmd5sum > /usr/tools/binary/testmd5
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 19:18:56       20 mac -rw-r--r-- root     root     /usr/tools/binary/testmd5
[root@localhost mac_daddy]# /usr/bin/md5sum /usr/tools/binary/testmd5
>/usr/tools/binary/testmd5.md5
[root@localhost mac_daddy]# ./mac_daddy.pl /usr/tools/binary/ | egrep test
Sep 16 2002 19:18:56       20 m.c -rw-r--r--  root     root     /usr/tools/binary/testmd5
Sep 16 2002 19:20:29       20 .a. -rw-r--r--  root     root     /usr/tools/binary/testmd5
                           60 mac -rw-r--r-- root     root     /usr/tools/binary/testmd5.md5
```

The only time that is changed from the */usr/bin/md5sum* utility is the access time. Since the */usr/bin/md5sum* utility does not have an option to save the information to a text file the person who ran the */usr/bin/md5sum* utility redirected the output to a file. This MACTimes of this file are all the same.

Below is a chart that shows the effects of the different process on the MACtimes. We will us it to find out in what command were executed on the binary and in what order.

	Modify Time	Access Time	Change Time
Executing Binary		X	
mv			X
cp (Original File)		X	

cp (New File)	X	X	X
zip/unzip			X
md5sum		X	
chmod			X

The only time that the modification time and access time would be the same would be if the person ran the *cp* command and used the new file to include in *sn.zip*. Below is a timeline analysis of what happened:

Time	Commands executed
n/a **	sn [Interface]
Apr 11 2002 09:29:52	md5sum sn >>sn.md5
Apr 11 2002 09:29:58	cp sn. sn.dat
n/a	zip –r sn sn.dat sn.md5

**This may or may not have happened. There is no way to prove it since the file was not acquired in a forensically correct manner and there is no other evidence (i.e. The_l0gz file) to support the assumption that the binary was executed.

The only command that we can't put into the timeline is */bin/chmod*. The only thing that we know is that */bin/chmod* was run before the binary was zipped. There is only six seconds between when the md5 checksum was calculated and the file name was changed, but if the person was quick enough or they issued multiple commands at one time (i.e. separating commands with ";") they could have changed the permissions within this amount of time.

The file *sn.dat* is a renamed copy of *ADMsniff*. We can see this from running *sn.dat* without any options. I was able to find a copy of *ADMsniff* and compile it to make the same executable. This will be shown later in the practical.

```
[root@localhost binary]# ./sn.dat
ADMsniff priv 1.0 <device> [HEADERSIZE] [DEBUG]
ex   : admsniff le0
 ..ooOO The ADM Crew OOoo..
```

While this is not enough evidence to prove that the file is really a version of *ADMsniff* we will run */usr/bin/strings –a* on *sn.dat* and see what other ASCII related information we can see. By looking at the information below we can see more evidence that *sn.dat* could be a copy of *ADMsniff.*

```
--=[ %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)
```

ADMsniff is a “libpap-based sniffer that is designed to be portable and powerful” according to the *README* file that is distributed with it. This sniffer only listens to traffic on TCP ports 21 (FTP Control), 23 (Telnet), 109(POP2), 110(POP3), 143(IMAP), 512(exec), 513(login), 514(shell), 1521(Oracle SQL), 31337(Many different backdoors). This is seen in the source code of *thesniff.*c.

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

There are three arguments that can be passed to *ADMsniff*. The only argument that is required is the <*device*> name (as seen in the source code).

```
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);
   }
```

When *ADMsniff* sees a packet that matches any one ports listed above it will save the data to the file *The_l0gz.* The information in *The_l0gz* shows the data portion of the packet in clear text. This information will usually be username and password. Depending on if the compromised host is on a switched network or in a hub environment, the attacker will have varying degrees of information.

From the information that we looked at in the “Binary Details” portion of the document we see that there is no way for us to know what time the binary was last executed.

For a step-by-step analysis of what *sn.dat* does when it is run we will look at the source code (thesniff.c from ADMsniff), some output from */usr/bin/strace -x -o strace.sn.dat -s 2000*, */usr/sbin/lsof -p*. The first operation that we will perform is */usr/sbin/lsof-p* on *sn.dat* :

```
[root@localhost binary]# ./sn.dat eth0 &
[1] 1434
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
[root@localhost binary]# lsof -p 1434
COMMAND  PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
sn.dat  1434 root  cwd    DIR    8,5   4096 342466 /usr/tools
sn.dat  1434 root  rtd    DIR    8,5   4096      2 /
sn.dat  1434 root  txt    REG    8,5 399124 342485 /usr/tools/sn.dat
sn.dat  1434 root    0u   CHR  136,0           2 /dev/pts/0
sn.dat  1434 root    1u   CHR  136,0           2 /dev/pts/0
sn.dat  1434 root    2u   CHR  136,0           2 /dev/pts/0
sn.dat  1434 root    3u  sock    0,0        3004 can't identify protocol
sn.dat  1434 root    4w   REG    8,5      0 342899 /usr/tools/The_l0gz
```

From the above output we can see that *sn.dat* opens up a regular file call *The_l0gz* in the same directory that the binary is in and it opens up a socket connection. This file is an ASCII text file that contains information outputted from *sn.dat*.

To see what steps the binary takes when executed we will trace the system calls with */usr/bin/strace -x -o strace.sn.dat -s 2000.*

```
[root@localhost binary]# strace -x -o strace.sn.dat -s 2000./sn.dat eth0
```

This will save the output from */usr/bin/*strace to a file called *strace.sn.dat*. We will then examine the contents of *strace.sn.dat* for information on how *sn.dat* operates. The binary is executed with the argument *eth0.*

```
execve("./sn.dat", ["./sn.dat", "eth0"], [/* 22 vars */]) = 0
```

The fcntl64 function if getting information about the first three file descriptors (0,1,2) with the *F_GETFD.*

```
fcntl64(0, F_GETFD)             = 0
fcntl64(1, F_GETFD)             = 0
fcntl64(2, F_GETFD)             = 0
```

The *uname* function is getting information about the current system.

```
uname({sys="Linux", node="localhost.localdomain", ...}) = 0
```

The next four functions are getting permissions information about the process. The first one is getting information about the effective UID (user ID) of the calling process. The second one is getting information about the real UID of the calling process. The third one is getting information about the effective GID (group ID) of the calling process. The fourth on is getting information about the real GID of the calling process.

```
geteuid32()                = 0
getuid32()                 = 0
getegid32()                = 0
getgid32()                 = 0
```

The *brk* syscall is returning the address of the end of the data segment.

```
brk(0)                     = 0x80ab488
brk(0x80ab4a8)             = 0x80ab4a8
brk(0x80ac000)                   = 0x80ac000
```

The binary is creating a socket and directly accessing the data link layer.

```
socket(PF_INET, SOCK_PACKET, 0x300 /* IPPROTO_??? */) = 3
```

The socket is then bound to the interface.

```
bind(3, {sin_family=AF_INET, sin_port=htons(25972), sin_addr=inet_addr("104.48.0.0")}},
16) = 0
```

The binary then gets the hardware address, MTU and interface flags of the interface that it is monitoring. The MTU is the maximum transmission unit that an IP datagram can send without fragmenting the packet. The reason that it is getting the MTU of the interface is so that it can set up a buffer that will be able to capture the entire packet. According to the code within *pcap-linux.c*[ii] the buffer will be set to 64 bytes bigger than the MTU to make sure that the link layer

[ii] *pcap-linux.c* as part libpcap version 0.4 as packaged with *ADMsniff*.

```
/* Base the buffer size on the interface MTU */
        memset(&ifr, 0, sizeof(ifr));
        strncpy(ifr.ifr_name, device, sizeof(ifr.ifr_name));
        if (ioctl(p->fd, SIOCGIFMTU, &ifr) < 0 ) {
                sprintf(ebuf, "SIOCGIFMTU: %s", pcap_strerror(errno));
                goto bad;
        }
/* Leave room for link header (which is never large under linux...) */
        p->bufsize = ifr.ifr_mtu + 64;
```

information is captured.. This way it will be able to capture all of the packet, including the data-link layer information.

```
ioctl(3, SIOCGIFHWADDR, 0xbffff890)     = 0
ioctl(3, SIOCGIFMTU, 0xbffff890)        = 0
ioctl(3, SIOCGIFFLAGS, 0xbffff890)      = 0
ioctl(3, SIOCSIFFLAGS, 0xbffff890)      = 0
```

The binary is checking the status of the file using the *fstat64* function and utilizing the *mmap* function to write the buffer to both memory and the file, *The_l0gz*, simultaneously. Since the file *The_l0gz* has not been created yet it is only preparing the buffer in memory.

```
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40000000
```

The binary is writing to the file descriptor (*STDOUT*) the information that we see when we run the binary.

```
write(1, "ADMsniff priv 1.0  in libpcap we trust !\n", 41) = 41
write(1, "credits: ADM, mel , ^pretty^ for the mail she sent me\n", 54) = 54
brk(0x80ad000)                          = 0x80ad000
```

The binary is opening the file *The_L0gz* in write only mode (*O_WRONLY*). The *0_CREAT* option tells the OS to create the file if it does not exist. The *O_TRUNC* option tells the OS to set the file size to zero if the file all ready exists and is a regular file. The permissions on this file are going to be se to 0666.

```
open("The_l0gz", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
```

We also see the sniffer in action capturing packets:

```
recvfrom(3,
"\x00\x50\x56\xf2\x4b\x3c\x00\x50\x56\xc0\x00\x01\x08\x00\x45\x00\x00\x38\x58\x5b\x0
0\x00\x80\x01\x62\x96\xc0\xa8\xff\x0
1\xc0\xa8\xff\x80\x03\x03\x50\xf1\x00\x00\x00\x00\x45\x00\x00\x48\x50\x2c\x40\x00\x40\
x11\x6a\xa5\xc0\xa8\xff\x80\xc0\xa8\xff\x01\x0
4\x03\x00\x35\x00\x34\xa7\x9f", 1564, 0, {sin_family=AF_UNIX, path="eth0"}, [18]) = 70
ioctl(3, 0x8906, 0xbffff970)            = 0
```

This will continue until a packet is captured that meets the port requirement. Then the binary will time stamp the packet.

```
recvfrom(3,
"\x00\x50\x56\xc0\x00\x01\x00\x50\x56\xf2\x4b\x3c\x08\x00\x45\x00\x00\x3c\x0f\xa5\x40
\x00\x40\x06\xab\x43\xc0\xa8\xff\x8
```

```
0\xc0\xa8\xff\x01\x04\x07\x00\x15\x27\xbf\x5f\x89\x00\x00\x00\x00\xa0\x02\x16\xd0\xd5\
xc9\x00\x00\x02\x04\x05\xb4\x04\x02\x08\x0a\x0
0\x09\x50\x2c\x00\x00\x00\x00\x01\x03\x03\x00", 1564, 0, {sin_family=AF_UNIX,
path="eth0"}, [18]) = 74
ioctl(3, 0x8906, 0xbffff970)          = 0
brk(0x80ae000)                      = 0x80ae000
time(NULL)                        = 1026998727
time(NULL)                        = 1026998727
time(NULL)                        = 1026998727
```

This packet is a packet for a FTP session. When it is about the write the packet to *The_l0gz* the output from */usr/bin/strace* will show the following:

```
fstat64(4, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40001000
write(4, "\n--=[ 192.168.255.1:21 --> 192.168.255.128:1031 ]=--
\n.....%)..P,220 3Com 3CDaemon FTP Server Ve
rsion 2.0.....%Q..Q.331 User name ok, need password.....%z..S+230 User logged
in.....%z..S,215 UNIX Type: L8.....%...S.2
00 Type set to I.....%...T.221 Service closing control connection.....%...T..\n", 337) =
337
```

We can see the *fstat64* system call retrieving information about the file and the *old_mmap* system call writing the information to memory. At the same time that the information is being written to memory it is also being written to *The_l0gz*. This cycle will continue until the binary is terminated.

Forensic Details

To see what footprints are left by this binary we will look at the output from */usr/sbin/lsof* and explain the use of inode information to see possible out of place creation times.

By running */usr/sbin/lsof* on the binary while it is running we can see any files that are used by the binary.

```
[root@localhost binary]# ./sn.dat eth0 &
[1] 1434
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
[root@localhost binary]# lsof -p 1434
COMMAND  PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
sn.dat  1434 root  cwd    DIR    8,5   4096 342466 /usr/tools
sn.dat  1434 root  rtd    DIR    8,5   4096      2 /
sn.dat  1434 root  txt    REG    8,5 399124 342485 /usr/tools/sn.dat
sn.dat  1434 root    0u   CHR  136,0            2 /dev/pts/0
sn.dat  1434 root    1u   CHR  136,0            2 /dev/pts/0
sn.dat  1434 root    2u   CHR  136,0            2 /dev/pts/0
sn.dat  1434 root    3u  sock    0,0         3004 can't identify protocol
```

```
sn.dat  1434 root    4w   REG    8,5      0 342899 /usr/tools/The_l0gz
```

We can see that it has a socket connection open and a regular file, called *The_L0gz*, open as well. Depending on how busy the system is there may be information that could be gained from the inode number. Since the binary is a sniffer it will set the *PROMISC* flag on the interface. This will show up when the system administrator reboots the system or if he/she runs */sbin/ifconfig*. The interface will show that it is in *PROMISC* mode. This information also shows up in */var/log/messages*. This is a good indicator that a sniffer is running or has been run.

```
[root@localhost tools]# tail -3 /var/log/messages
Oct 23 06:03:19 localhost kernel: sn.dat uses obsolete (PF_INET,SOCK_PACKET)
Oct 23 06:03:19 localhost kernel: eth0: Promiscuous mode enabled.
Oct 23 06:03:19 localhost kernel: device eth0 entered promiscuous mode
[root@localhost tools]#
```

From the above output we can see that *sn.dat* uses on old socket call and sets the interface to promiscuous mode. Depending on where the attacker hid this binary the MACtimes might show something suspicious. If the binary is hidden in a low activity area it would show up, but if it hidden in a highly accessed area then it will get hidden in normal activity. Depending on the activity of the server that was compromised the inode information might give away some clues. The inode number will be rather new (higher). Depending on when the binary was installed by the attacker and when the server was forensically analyzed this may show up. If there were a lot of files created after the binary was installed then the inode number might not stand out. This also depends on where the binary was installed. If the binary was installed in a directory that is usually static (i.e. /dev) the new inode number will stick out. If the attacker installed the binary in a directory that has a lot of files created in it regularly then the inode number might not show up as easily. When a hacker breaks into a system he/she will usually attempt to hide the binary in a directory that has a lot of files in it. Attacker's also tries to name their binaries to something that will not be so easily noticed.

Program Identification

From executing */usr/bin/strings –a* on *sn.dat* we saw right away the following information that helps us find the source code and what it was complied with:

```
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
priv 1.0
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
/USR/BIN/GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98).
```

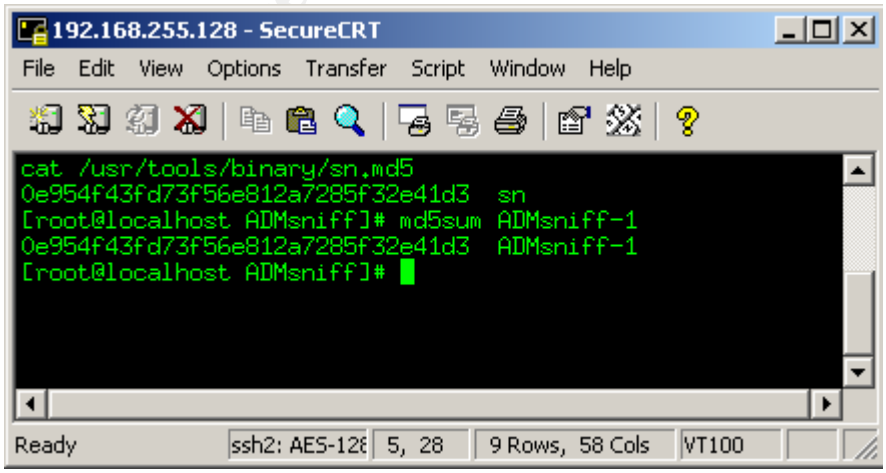
A search on the Internet using the google search engine found many copies of *ADMsniff-v.08* but only a couple of sites had *ADMsniff priv 1.0.* The file that I downloaded was *ADMsniff.tgz.* The version of */USR/BIN/GCC* that the attacker used to compile *ADMsniff* is the same that is shipped with Red Hat Linux 7.2. The *Makefile* that is included with this binary does not create a statically linked executable so I edited the *Makefile* and added the *–static* keyword so that the file that I created would be statically linked. When I first attempted to compile the binary I received an error:

```
[root@localhost ADMsniff]# gcc thesniff.c -static -lpcap -oADMsniff-1
/usr/bin/ld: cannot find -lpcap
collect2: ld returned 1 exit status
[root@localhost ADMsniff]#
```

After looking this error up I found out that the linker, *ld*, only looks for libraries in certain directories. I place the *libpcap.a* library in the */usr/lib* directory and tried to compile it again. This time the binary successfully compiled. I then fully stripped the binary using */usr/bin/strip*.

```
[root@localhost ADMsniff]# gcc thesniff.c -static -lpcap -oADMsniff-1
[root@localhost ADMsniff]# /usr/bin/strip ADMsniff-1
[root@localhost ADMsniff]#
```

After compiling the binary and stripping it I ran */bin/ls –la ADMsniff-1* and saw that the file was 399,124 bytes. After this I ran */usr/bin/md5sum ADMsniff-1* and compared it to the md5 checksum of *sn* and it was the same.



When running */usr/sbin/lsof –p* on both of the binaries (*sn* and *ADMsniff-1*) it shows that they both access the same file, *The_l0gz*, and they both open a socket connection. We will look at all the syscalls made my *ADMsniff-1* and

compare them to the syscalls made *sn.dat.* To find this information we will use */usr/bin/objdump –D*. This will dump the files contents into assembly language format. Looking at the assembly language we need to find *int $0x80* or "*cd 80*" in hexadecimal. This is the assembly language instruction for operating system calls. Once we find this we look for the *eax* register's value just before the *int $0x80* and this will tell us the value of the system call being made. We will then cross reference this with the file */usr/include/asm/unistd.h* to get the name of the syscall's being executed. To simplify the cross referencing process I wrote a PERL script.

```
[root@localhost ADMsniff]# ./syscall_finder.pl ADMsniff-1
sh: obdump: command not found
Total entries: 42
Call: 01        #define __NR_exit              1
Call: 03        #define __NR_read               3
Call: 04        #define __NR_write              4
Call: 05        #define __NR_open               5
Call: 06        #define __NR_close              6
Call: 0d        #define __NR_time              13
Call: 13        #define __NR_lseek             19
Call: 14        #define __NR_getpid            20
Call: 18        #define __NR_getuid            24
Call: 21        #define __NR_access            33
Call: 25        #define __NR_kill              37
Call: 2f        #define __NR_getgid            47
Call: 31        #define __NR_geteuid           49
Call: 32        #define __NR_getegid           50
Call: 36        #define __NR_ioctl             54
Call: 37        #define __NR_fcntl             55
Call: 37        #define __NR_fcntl             55
Call: 37        #define __NR_fcntl             55
Call: 4c        #define __NR_getrlimit         76 /* Back compatible 2Gig limited rlimit */
Call: 55        #define __NR_readlink          85
Call: 5b        #define __NR_munmap            91
Call: 68        #define __NR_setitimer        104
Call: 6a        #define __NR_stat             106
Call: 6c        #define __NR_fstat            108
Call: 77        #define __NR_sigreturn        119
Call: 7a        #define __NR_uname            122
Call: 7d        #define __NR_mprotect         125
Call: 8c        #define __NR__llseek          140
Call: 92        #define __NR_writev           146
Call: a3        #define __NR_mremap           163
Call: ad        #define __NR_rt_sigreturn     173
Call: ae        #define __NR_rt_sigaction     174
Call: af        #define __NR_rt_sigprocmask   175
Call: b7        #define __NR_getcwd           183
Call: bf        #define __NR_ugetrlimit             191  /* SuS compliant getrlimit */
Call: c3        #define __NR_stat64           195
Call: c5        #define __NR_fstat64          197
Call: c7        #define __NR_getuid32         199
```

```
Call: c8        #define __NR_getgid32         200
Call: c9        #define __NR_geteuid32        201
Call: ca        #define __NR_getegid32        202
Call: dd        #define __NR_fcntl64          221
[root@localhost ADMsniff]# ./syscall_finder.pl /usr/tools/binary/sn.dat
sh: obdump: command not found
Total entries: 42
Call: 01        #define __NR_exit             1
Call: 03        #define __NR_read             3
Call: 04        #define __NR_write            4
Call: 05        #define __NR_open             5
Call: 06        #define __NR_close            6
Call: 0d        #define __NR_time             13
Call: 13        #define __NR_lseek            19
Call: 14        #define __NR_getpid           20
Call: 18        #define __NR_getuid           24
Call: 21        #define __NR_access           33
Call: 25        #define __NR_kill             37
Call: 2f        #define __NR_getgid           47
Call: 31        #define __NR_geteuid          49
Call: 32        #define __NR_getegid          50
Call: 36        #define __NR_ioctl            54
Call: 37        #define __NR_fcntl            55
Call: 37        #define __NR_fcntl            55
Call: 37        #define __NR_fcntl            55
Call: 4c        #define __NR_getrlimit        76    /* Back compatible 2Gig limited rlimit */
Call: 55        #define __NR_readlink         85
Call: 5b        #define __NR_munmap           91
Call: 68        #define __NR_setitimer        104
Call: 6a        #define __NR_stat             106
Call: 6c        #define __NR_fstat            108
Call: 77        #define __NR_sigreturn        119
Call: 7a        #define __NR_uname            122
Call: 7d        #define __NR_mprotect         125
Call: 8c        #define __NR__llseek          140
Call: 92        #define __NR_writev           146
Call: a3        #define __NR_mremap           163
Call: ad        #define __NR_rt_sigreturn     173
Call: ae        #define __NR_rt_sigaction     174
Call: af        #define __NR_rt_sigprocmask   175
Call: b7        #define __NR_getcwd           183
Call: bf        #define __NR_ugetrlimit       191    /* SuS compliant getrlimit */
Call: c3        #define __NR_stat64           195
Call: c5        #define __NR_fstat64          197
Call: c7        #define __NR_getuid32         199
Call: c8        #define __NR_getgid32         200
Call: c9        #define __NR_geteuid32        201
Call: ca        #define __NR_getegid32        202
Call: dd        #define __NR_fcntl64          221
```

For a quick verification of the two files (*ADMsniff-1* and *sn*) syscall's I sent the output from the two different files to separate output files and ran */usr/bin/wc –l* and */usr/bin/md5sum* on both files to see if they matched.

```
[root@localhost ADMsniff]# ./syscall_finder.pl ADMsniff-1 >sysadm
sh: obdump: command not found
[root@localhost ADMsniff]# ./syscall_finder.pl /usr/tools/binary/sn.dat >syssn
sh: obdump: command not found
[root@localhost ADMsniff]# wc -l syssn
    43 syssn
[root@localhost ADMsniff]# wc -l sysadm
    43 sysadm
[root@localhost ADMsniff]# /usr/bin/md5sum sysadm
2d4f29ea7e00ba056a9d2ed4ffd8265b  sysadm
[root@localhost ADMsniff]# /usr/bin/md5sum syssn
2d4f29ea7e00ba056a9d2ed4ffd8265b  syssn
```

From the output above we can see that they two binaries use the same syscalls (amount and placement/order). With all the evidence that we have we can say with 100% accuracy that the binary that we reverse engineered was a copy of *ADMsniff priv 1.0*.

Legal Implications

With the MACtime analysis that was conducted earlier I was not able to prove that the file was actually executed. I was only able to prove three facts:

1. The original file name is *sn*.
2. The */usr/bin/md5sum* utility was run on *sn* at Apr 11 2002 09:29:52.
3. The file *sn* was copied to *sn.dat* at Apr 11 2002 09:29:58.
4. The files *sn.dat* and *sn.md5* were zipped.

Our company policy prohibits any user to be in possession of any tools or utilities that would be considered malicious or a "hacker" tool. The only departments that are allowed to own, or have possession of, a sniffer is network engineering and information security. The sniffer applications that are authorized for use within the company are Network General Sniffer's DSS (Distributed Sniffer System), EtherPeek, and Ethereal. If any person, including anyone in network engineering or information security, is found to be in possession of a sniffer that has not been authorized he/she will be disciplined accordingly.

The sniffer that was found on the system was not a sniffer that could be used for any troubleshooting purpose or investigation. This sniffer does not allow the user to do the following things that are common among other sniffers:

1. Filter the traffic. The only way that the user can modify the filtering process is to modify the *coolport* array of *thesniff.c* and recompile *thesniff.c*.
2. Ability to decode traffic. Most, if not all, sniffers not only capture packets but can also decode them. Regular sniffers are able to decode not only TCP packets but others as well (i.e. UDP, other IP protocols, IPX, Apple Talk).
3. Ability to see raw packet. Sniffers usually allow people to view the packets in different ways. Most of the time the users will view the packet in a "decode" view, but they also have other views that they can view the captured packets.

The only purpose of this sniffer is to capture username's and password's and some user data. Interestingly enough our company's acceptable network policy states that any electronic transmission of data can be intercepted or monitored with out prior consent of the parties being monitored. There is no stipulation on who can and can't do the monitoring or who has to authorize it either. The only stipulation is that the information can not be used for personal gain. However the likelihood of someone possessing this utility and not using for personal gain is highly unlikely.

For us to use the binary as part of evidence we would need to have some more information, like the *The_l0gz* file, or an image of the directory where the binary was found. If the binary had been collected in a forensically sound manner (i.e. *dd*) the binary would give us more clues, but we would still need other pieces of evidence.

The use of this tool on the system would violate the Wiretap Act since a certain amount of the content of the packet is being kept. Since this utility does not posses any troubleshooting functionality it has no use in any toolkit for either network engineering or information security.

Interview Questions

The following questions would have been asked if we were able to determine who attacked and compromised the server. From my experience the evidence had a "look and feel" of a technical exam. The evidence was like nothing that I have encountered in my professional work experience or working with the South Florida Honeynet Project. It did seem very similar to the way that high level technical certifications tests are carried out though. Even though I don't believe that an attacker left the binary on the system I still came up with five questions for the "attacker".

Question 1: Are you familiar with the Linux operating system?
Reason: I want to establish if the attack has or has not used the Linux operating system.

Question 2: Have to worked with any types of sniffers, like tcpdump,snoop, or ethereal?
Reason: This may help establish that the attacker's level of knowledge about how sniffers operate and what type of information they can get.

Question 3. Do you do any program development in the Unix/Linux environment?
Reason: The attacker had to have some knowledge to know to move the *libpcap.a* library to the a directory where the linker would look for it prior to compiling the binary. The attacker also had to know about statically compiling the binary.

Question 4: Have you ever heard of a program called *ADMsniff*?
Reason: This will let us know if the attacker knows what this program does.

Question 5: Do you like challenges?
Reason: If the attacker does not like challenges then he/she may not be the real attacker. If the attacker does like challenges it shows that they make like hacking and consider it a game.

Additional Information

To assist with the reverse engineering of the binary I used the site http://linuxassembly.org/linasm.html. This site provided me the information to correlate the output from */usr/bin/objdump –D* to */usr/include/asm/unistd.h*. This allowed me to find out what system calls were made by each binary. I automated this information by writing a PERL script, *syscall_finder.pl*. For more information on reverse engineering I used http://www.hut.fi/~kalyytik/hacker/ssh-crc32-exploit_Korpinen_Lyytikainen.html and http://www.hut.fi/~kalyytik/hacker/security_breach_test_report_Korpinen_Lyytikainen.html. These are excellent papers on the reverse engineering process of the X2 exploit from a developers prospective. For information about assembly code translation specific to vulnerabilities I used http://lsd-pl.net/papers.html.

Part III

Legal Issues of Incident Handling

As a senior information security engineer I am called on to troubleshoot many different issues, and I often use a sniffer. All though there are different laws out there restricting the use of "network taps" they all have clauses that outline under what circumstances that a person can legally use a "network tap". Under the Wiretap Act, U.S.C 2511, clause (2)(a)(i), allows me to monitor network traffic legally for most of the situations that I encounter. This clause basically

states that as long as I am providing a service that is part of my job function then I can use a sniffer. In most cases I am not acting under color of the law so U.S.C. 2511(2)(c) does not apply, even though when I am troubleshooting an issue I have the consent of the user. The Wiretap Act also allows me to monitor traffic if I feel that there is a threat to any one of our systems. I would have to filter the traffic for the latter scenario.

The newly enacted Patriot Act enables system administrator more flexibility in using network taps, but is geared toward government agencies rather than the private sector. The Patriot Act uses the term "under color of law" in a lot of it's definitions. Like previously stated, when I use a sniffer it is part of my job function so I am able to do so freely with out violating the Wiretap Act and since I am not acting under the "color of law" I don't concern myself with the Patriot Act. What is unique in my situation is that I am not a system administrator. The functions that I am required to perform for the company allow me to monitor all network traffic, content included. Most system administrators might not use network monitoring tools as part of their job function so they would have to make sure that they are covered by either the Wiretap Act or the Patriot Act prior to monitoring traffic.

There are four distinct times when I would be called on to use a sniffer. I will outline each one of them and detail my responsibilities to the company and the user. In of these cases I would be exempt from prosecution because of U.S.C. 2511(2)(a)(i). One of them, situation three, could even be covered by the Pen Register Act as I am only looking at packet header information.

Situation One: Troubleshoot a user issue in which the user has already worked with other technical people within the company and the others were not able to troubleshot the issue. I usually troubleshoot all of these issues with the use of a sniffer. Using a sniffer offers a different view when troubleshooting an application that uses network resources. I don't have to worry about what the developer thinks should be sent from a client to server or visa versa. I can see exactly what is sent. When I use a sniffer I will most likely be capturing the entire packet and not just the header information

Situation Two: Troubleshoot an issue that effects many users and is thought to be an application issue. In this scenario I will work with the developers of the application instead of the users. I would usually capture all traffic going to a particular application and filter the packets out later. This allows me to see many different users data. It is not feasible to get the consent of all the users, but I can get the permission of the developers.

Situation Three: Troubleshoot an issue that is network related (i.e. duplex mismatching, bandwidth saturation). In this scenario I will be working either alone or with the network engineering department. I will be capturing a lot of traffic and be looking for any issues that would help determine the issue. In this case I can capture just the header information.

Situation Four: Investigating an active attack on the network.

In this scenario I would use a sniffer for forensic evidence and also to understand what the attacker did. I would be capturing the entire packet so that I could see what the attacker is doing without any type of translation. Looking at log files on a system can provide some type of evidence but if does not understand the request that was sent it may not log it properly.

Just because I am part of the information security staff doesn't mean that I have the right to monitor network traffic without a legitimate reason. If the reason for me to monitor the network does not follow any of the aforementioned situations then I would be in violation of the Wiretap Act because it would not be part of my job function. If the situation does not involve troubleshooting or an actual network anomaly/attack I make sure that I get permission from a VP prior to capturing network traffic. While this does not take the responsibility away from me it does give me some more assurance about my reasoning for doing a network capture.

There are some instances were I would be abiding by corporate policy, but not be abiding by US law. If the company were to want me to capture all traffic from a user because the user might be doing something illegal I would be following company policy but not be following USC 2511 (2)(a)(i). Unless the user is actually doing something malicious at the time that I want to capture the traffic, I can't monitor all their network traffic hoping to capture something. The company would need a court order for this type of monitoring to be carried out.

As a part of the information security staff we can set a company policy that all systems that offer particular services have a banner on them, but we do not actually touch the systems to put the banner in place. Since we run an all Microsoft client network we have the policy set so that each system has a logon banner. This is set in "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system\" with a string value name of "legalnoticetext" and an appropriate legal notice in the banner. This notice will show up on any PC that is running a version of Microsoft Windows (greater than 3.11) that is set up to use the "Client for Microsoft Networks". The "Client for Microsoft Networks" needs to be set to log onto the appropriate NT domain. This will allow us to notify any user that their activity on the network could be monitored and that by using the system they are consenting to the regulations. This would suffice if the person logged onto the network from a PC that is part of the domain, but what if they didn't? There are a few scenarios where someone could use internal resources without even seeing the logon banner.

Scenario One: Attacker uses a PC that was never logged off the network by the current user. This scenario would not be hard to do at the company that I work for. There are many people that view logging onto the network as a hassle and leave their PC's logged in at all times. It would not be hard to sit down at someone's PC that is logged into the network and access the resources that they have and also launching some attacks from the victim's PC. If an attacker were to do this they would not see the login banner at all. If we are not able to force

the users to be responsible for their user ID's and to secure their PC's when they are not at them there is little that we can do to persecute any attacker since it would be hard to prove who was sitting at the PC at the time of attack. If the company had video cameras all over then they would be able to catch the attacker, but this is not the case.

Scenario Two: Attacker used personal PC that is not part of the network or domain. This type of attack would also be easy to execute within the company. There is no policy that states that user's can't bring in a personal laptop into the company and plug it into the network. There is also no regulation from the physical security team on watching what employees bring into the company. The attack does not have to be an employee though. Like many companies if the attacker "looks the part" and enters the company during the busy times he/she has a high chance of getting into the company, unless the physical security group has some way to enforce the policy of making sure that all employees wear their badge and are identified prior to entering the building this type of a scenario is very easy to carry out.

Scenario Three: An attacker used the Internet to attack publicly available systems. This attack would be mainly focused on web, email or FTP servers. We can place a banner on the email and FTP servers, but not on the HTTP servers. The most targeted systems are usually the web server. There are a couple of reasons for this. The first is that they are running Microsoft's IIS. IIS has had a lot of security issues in the past and is a favorite among hackers. The second is that the attacker can not only gain access to the internals of the web server, but in a lot of cases the attacker will be able to attack any backend database servers.

Some services do not have any method of displaying a logon banner, like HTTP. These services were not meant for a user to connect to directly without the use of a specific client application. With HTTP a company can post a policy concerning the use of their web site and state that by using the site you consent to any type of monitoring, but this does not mean that the user saw the policy and gave their consent. Since most attackers use custom scripts/utilities to attack web servers they don't have any way of seeing the links to the security policy. Currently there exists no software that can ensure that the user has seen and acknowledged the websites security policy. The software would not be hard to implement. When an attacker creates a script/utility to attack a web servers he/she only needs to specify which HTTP version that he/she is using. One of the easiest ways to do a "banner grab" on a web server is to telnet to port 80 and type "get" and hit the "enter". This will give back an error like the one show below.

```
HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Mon, 26 Aug 2002 16:00:12 GMT
Content-Type: text/html
Content-Length: 87
```

<html><head><title>Error</title></head><body>The parameter is incorrect. </body></html>

Connection to host lost.

During this process I was not able to see any security policy. This is a similar view to the attack when he/she attempts to exploit a server. Depending on the situation a logon banner may or may not be of use. Since we are not a government agency there is more things that we can do without a court order. One of them is capturing the traffic from the attacker for evidence. Under USC 2511 (2)(a)(i) we would be able to capture the attacker's network traffic because we would be attempting to protect the property on the server being attacked. The network capture will not stop the attacker from getting any information, but it may be of assistance when it comes to finding out what the attacker was doing so that we could be more focused when it comes to the forensic investigation or any legal matters.

Last Updated: November 15th, 2018

Upcoming SANS Forensics Training

SANS Paris November 2018	Paris, France	Nov 19, 2018 - Nov 24, 2018	Live Event
SANS November Singapore 2018	Singapore, Singapore	Nov 19, 2018 - Nov 24, 2018	Live Event
SANS Stockholm 2018	Stockholm, Sweden	Nov 26, 2018 - Dec 01, 2018	Live Event
SANS San Francisco Fall 2018	San Francisco, CA	Nov 26, 2018 - Dec 01, 2018	Live Event
SANS Austin 2018	Austin, TX	Nov 26, 2018 - Dec 01, 2018	Live Event
SANS Khobar 2018	Khobar, Kingdom Of Saudi Arabia	Dec 01, 2018 - Dec 06, 2018	Live Event
SANS Nashville 2018	Nashville, TN	Dec 03, 2018 - Dec 08, 2018	Live Event
SANS Frankfurt 2018	Frankfurt, Germany	Dec 10, 2018 - Dec 15, 2018	Live Event
SANS Cyber Defense Initiative 2018	Washington, DC	Dec 11, 2018 - Dec 18, 2018	Live Event
Cyber Defense Initiative 2018 - FOR500: Windows Forensic Analysis	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Cyber Defense Initiative 2018 - FOR572: Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Cyber Defense Initiative 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Cyber Defense Initiative 2018 - FOR585: Advanced Smartphone Forensics	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Cyber Defense Initiative 2018 - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	Washington, DC	Dec 13, 2018 - Dec 18, 2018	vLive
Mentor Session - FOR500	Phoenix, AZ	Jan 11, 2019 - Feb 15, 2019	Mentor
SANS Threat Hunting London 2019	London, United Kingdom	Jan 14, 2019 - Jan 19, 2019	Live Event
SANS Amsterdam January 2019	Amsterdam, Netherlands	Jan 14, 2019 - Jan 19, 2019	Live Event
Mentor Session - FOR508	Copenhagen, Denmark	Jan 16, 2019 - Mar 09, 2019	Mentor
Cyber Threat Intelligence Summit & Training 2019	Arlington, VA	Jan 21, 2019 - Jan 28, 2019	Live Event
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	FOR610 - 201901,	Jan 21, 2019 - Feb 27, 2019	vLive
SANS Miami 2019	Miami, FL	Jan 21, 2019 - Jan 26, 2019	Live Event
Mentor Session - FOR585	Tampa, FL	Jan 24, 2019 - Mar 07, 2019	Mentor
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
Security East 2019 - FOR585: Advanced Smartphone Forensics	New Orleans, LA	Feb 04, 2019 - Feb 09, 2019	vLive
SANS London February 2019	London, United Kingdom	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Anaheim 2019	Anaheim, CA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS vLive - FOR578: Cyber Threat Intelligence	FOR578 - 201902,	Feb 11, 2019 - Mar 20, 2019	vLive
Community SANS Madrid FOR610 (in Spanish)	Madrid, Spain	Feb 11, 2019 - Feb 16, 2019	Community SANS
SANS Northern VA Spring- Tysons 2019	Vienna, VA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Dallas 2019	Dallas, TX	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS New York Metro Winter 2019	Jersey City, NJ	Feb 18, 2019 - Feb 23, 2019	Live Event