Global Information Assurance Certification Paper

Analysis of a Compromised Honeypot

Practical Assignment Version 1.4

26th July 2004

Stephen Hall GCIA GCIH

GIAC Certified Forensic Analyst (GCFA)
Practical Assignment

Part 1 - Analyze an Unknown Binary

Introduction

I have been called in to perform a *Computer Forensic Investigation* of a single floppy disk that has been recovered from the floppy disk drive of a computer system used by a company employee named John Price. Contained on this floppy disk is an unknown binary file named ***prog***. My primary task within this investigation is to identify the unknown binary. Following on from this, I will attempt to discover if John Price has been using the binary to the detriment of the company or for illegal means.

Preparation

Before I can commence an investigation, I must perform a number of steps to ensure that the investigation I am about to perform is done using equipment that is in a sound condition. Failure to perform these steps may put doubt on my investigation should it later result in court proceedings.

Verification of Forensic Workstation

Firstly, I must verify the installation of the forensic workstation. As my workstation is based on the Linux distribution Fedora Core 1, I can do this using the rpm –Va option as shown below. The output reports the following results for each file added to the system via the rpm package system:

S	file Size differs
M	Mode differs (includes permissions and file type)
5	MD5 sum differs
D	Device major/minor number mis-match
L	readLink path mis-match
U	User ownership differs
G	Group ownership differs
T	mTime differs

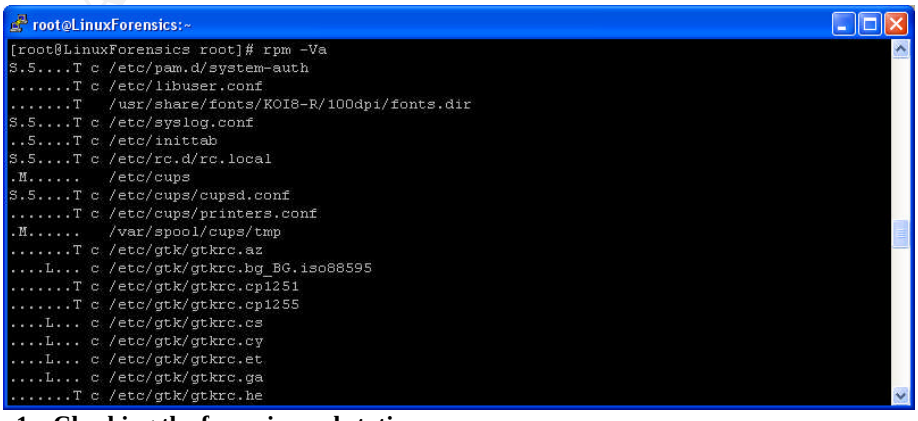


Figure 1 – Checking the forensic workstation

In the above example, I can see that some files have changed (i.e. their entries are not all "."), but these are known files, or directories which contain configuration details, or log files.

Cleansing of the forensic disk space

In addition to this step, the disk space which is to hold the evidence for this investigation must be cleansed to ensure that no information from a previous investigation could possibly remain on the media to contaminate the evidence.

This is performed by using the *dd* command, and the special Linux device /dev/zero. Once the filesystem has been cleansed then a new Linux filesystem has to be created, and the filesystem mounted ready for use. All evidence to be investigated on the system is to be held in the /evidence directory.

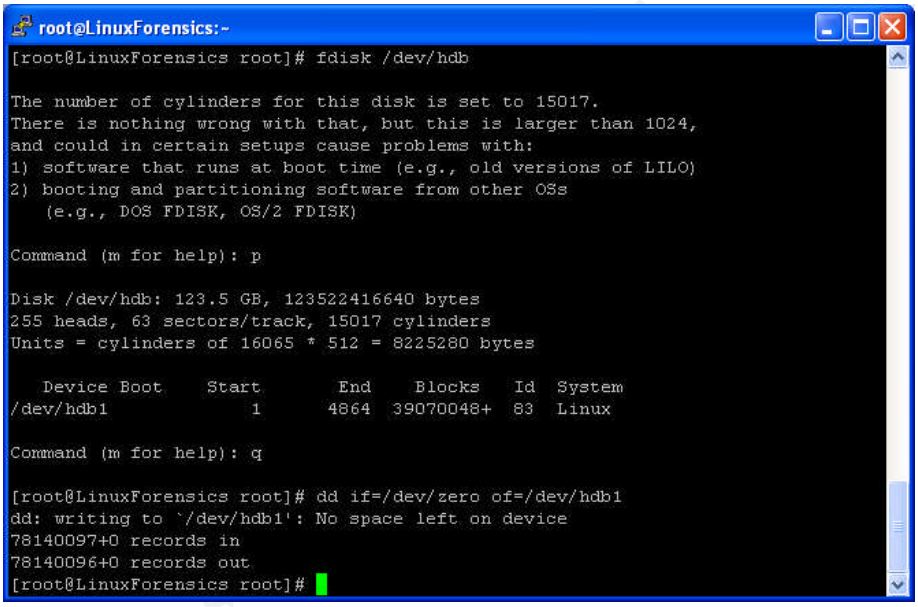


Figure 2 – Zeroing the forensic evidence disk space

Items of evidence

I have received a sealed and tagged evidence bag containing the item to be investigated. I sign for receipt of the evidence and enter the transfer of the item in the evidence log. The item entered into evidence consists of:

> A single 3.5 inch TDK floppy disk which has been sealed within a plastic bag and tagged with the reference: Tag# fl-160703-jp1

An image of this floppy disk has already been taken, and the following evidence entered:

Tag Number	Description	MD5 Hash	Image name
fl-160703-jp1	3.5 inch TDK floppy disk	4b680767a2aed974cec5fbcbf84cc97a	fl-160703-jp1.dd.gz

Table 1 – Evidence Log

To confirm the image I have is correct, I will verify the md5 checksum as shown in figure 3 below:

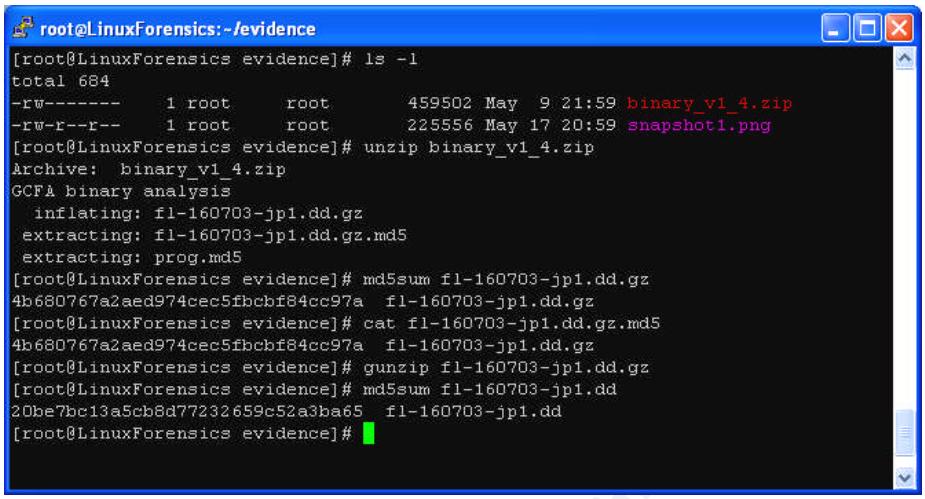


Figure 3 – Verification of the supplied image

By using the ***md5sum*** command I can confirm that the md5 checksum of the unzipped binary fl-160703-jp1.dd.gz is identical to the entry on the evidence log provided. In addition to this, an initial md5 checksum is immediately taken on the uncompressed image as this will be the image I shall use. This was recorded in the evidence log. The file prog.md5 is also supplied, I will later use this file to compare with the recovered binary.

Recovery of the unknown program

To investigate the image, the forensic analysis tool Autopsy version 2.0[1] in conjunction with Sleuthkit version 1.69[2] will be used.

Autopsy allows the forensic analysis to be performed both on the local forensic workstation, and remotely. In this investigation the analysis was performed remotely to allow the capture of screen images more easily. To achieve this, the following command was executed on the Forensic Workstation:

```
# ./autopsy -C -d /evidence -p 5555 192.168.1.103
```

The parameters passed to autopsy where:

-C	Tells autopsy not to put cookies in the URL
-d /evidence	Specifies the evidence locker to use
-p 5555	Specifies the local port number for Autopsy
192.168.1.103	Specifies the IP address of the system allowed to use autopsy

[1] http://www.sleuthkit.org/autopsy/index.php

[2] http://www.sleuthkit.org/sleuthkit/index.php

Where files have been exported, this was achieved by setting the IP address to localhost and using a browser on the forensic workstation.

Utilising Autopsy 2.0 I can quickly export the *prog* binary for analysis:

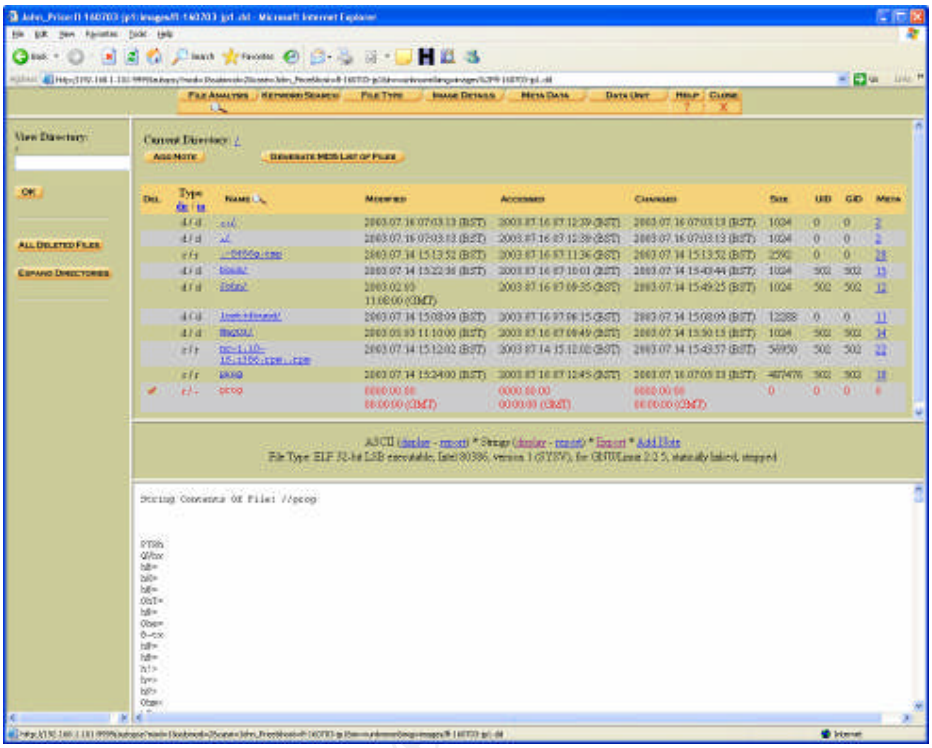


Figure 4 – Autopsy File Analysis showing *Export* option (example)

To discover what this unknown binary is, I need to look at it more closely. Using the autopsy "file analysis" feature as shown in Figure 4, I am able to select the binary and export this to my forensic workstation by selecting the export option.

To confirm this binary is the correct one, I can compare it against the known md5 supplied with the forensic image, thus:

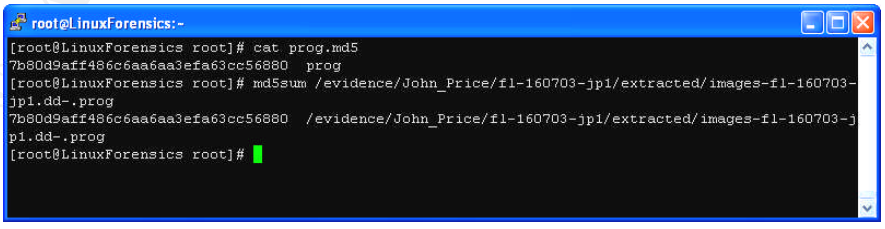


Figure 5 – Comparison of the prog.md5 value and the md5 of the extracted binary

Forensic Details

As the md5sum values match, I can safely carry on with the investigation. Initial analysis of the file shows us the following:

```
[root@LinuxForensics extracted]# file images-fl-160703-jp1.dd-.prog
images-fl-160703-jp1.dd-.prog: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), for GNU/Linux 2.2.5, statically linked, stripped
[root@LinuxForensics extracted]#
```

Figure 6 – Initial file analysis using the *file* command

From this I can learn the following:

- The code is in ELF format
- It is for a 32 bit, little Endian platform
- It is compiled for an Intel platform
- It has been statically linked
- It has been stripped of any symbol table
- It was compiled using a Linux 2.2.5 system

As I do not yet know what this file is, what it does, or where it was from, I cannot trust it. Therefore to safely execute the binary, I need to host it in a sacrificial machine. I utilise the Microsoft Virtual PC 2004 virtual machine environment to perform this task. As shown below, I safely execute the file, gaining more information about the function of the binary.

```
[root@localhost evidence]# ./images-fl-160703-jp1.dd-.prog --help
prog:1.0.20 (07/15/03) newt
Usage: prog [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help  display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  m  list sector numbers
  c  extract a copy from the raw device
  s  display data
  p  place data
  w  wipe
  chk  test (returns 0 if exist)
  sb  print number of bytes available
  wipe  wipe the file from the raw device
  frag  display fragmentation information for the file
  checkfrag  test for fragmentation (returns 0 if file is fragmented)
--outfile <filename> write output to ...
--label useless bogus option
--name  useless bogus option
--verbose       be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit> logging
threshold ...
--target <filename> operate on ...
[root@localhost evidence]#
```

Figure 7 – Initial execution of the unknown binary

As I can now safely execute the program, I need to perform some analysis to see if the program affects the system it is run upon. Firstly, I need to check if

any files are used when the program is executed. I can use the *strace* function to verify this:

```
 [root@localhost evidence]# md5sum images-fl-160703-jp1.dd-.prog
7b80d9aff486c6aa6aa3efa63cc56880  images-fl-160703-jp1.dd-.prog
[root@localhost evidence]# strace -fvx ./mages-fl-160703-jp1.dd-.prog
execve("./images-fl-160703-jp1.dd-.prog", ["./images-fl-160703-jp1.dd-.prog"], [/* 22
vars */]) = 0
fcntl64(0, F_GETFD)                     = 0
fcntl64(1, F_GETFD)                     = 0
fcntl64(2, F_GETFD)                     = 0
uname({sysname="Linux", nodename="localhost.localdomain", release="2.4.18-3",
version="#1 Thu Apr 18 07:32:41 EDT 2002", machine="i686"}) = 0
geteuid32()                             = 0
getuid32()                              = 0
getegid32()                             = 0
getgid32()                              = 0
brk(0)                                  = 0x80bedec
brk(0x80bee0c)                          = 0x80bee0c
brk(0x80bf000)                          = 0x80bf000
brk(0x80c0000)                          = 0x80c0000
write(2, "no filename. try \'--help\' for he"..., 36no filename. try '--help' for
help.
) = 36
_exit(2)                                = ?
[root@localhost evidence]#
```

Figure 8 – strace output from execution of unknown binary

I have used the following options for strace:

- -f Follow forks
- -v Be verbose in the output of environment calls etc.
- -x Print all none ASCII characters in hex character format

As can be seen from figure 9, the simple execution of the binary does not show any unusual impact being exhibited on the host system.

I now need to execute the unknown binary to see how it interacts with the Linux file system. I use the same options for strace as before, except I now pass more parameters to the unknown binary. As you can see below, again the unknown binary does not use any other program, but does access the file specified (testfile), and the raw file system that contains for testfile (/dev/hda2):

```
[root@localhost evidence]# touch testfile
[root@localhost evidence]# strace -fvx ./images-fl-160703-jp1.dd-.prog --mode chk
testfile
execve("./images-fl-160703-jp1.dd-.prog", ["./images-fl-160703-jp1.dd-.prog", "--
mode", "chk", "testfile"], [/* 22 vars */]) = 0
fcntl64(0, F_GETFD)                     = 0
fcntl64(1, F_GETFD)                     = 0
fcntl64(2, F_GETFD)                     = 0
uname({sysname="Linux", nodename="localhost.localdomain", release="2.4.18-3",
version="#1 Thu Apr 18 07:32:41 EDT 2002", machine="i686"}) = 0
geteuid32()                             = 0
getuid32()                              = 0
getegid32()                             = 0
getgid32()                              = 0
brk(0)                                  = 0x80bedec
brk(0x80bee0c)                          = 0x80bee0c
brk(0x80bf000)                          = 0x80bf000
brk(0x80c0000)                          = 0x80c0000
lstat64("testfile", {st_dev=makedev(3, 2), st_ino=1488809, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=0, st_size=0,
```

```
st_atime=2004/06/26-20:58:57, st_mtime=2004/06/26-20:58:57, st_ctime=2004/06/26-
20:58:57}) = 0
open("testfile", O_RDONLY|O_LARGEFILE)  = 3
ioctl(3, FIGETBSZ, 0xbffff964)          = 0
lstat64("testfile", {st_dev=makedev(3, 2), st_ino=1488809, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=0, st_size=0,
st_atime=2004/06/26-20:58:57, st_mtime=2004/06/26-20:58:57, st_ctime=2004/06/26-
20:58:57}) = 0
lstat64("/dev/hda2", {st_dev=makedev(3, 2), st_ino=65880, st_mode=S_IFBLK|0660,
st_nlink=1, st_uid=0, st_gid=6, st_blksize=4096, st_blocks=0, st_rdev=makedev(3, 2),
st_atime=2002/04/11-15:25:14, st_mtime=2002/04/11-15:25:14, st_ctime=2004/05/23-
22:51:31}) = 0
open("/dev/hda2", O_RDONLY|O_LARGEFILE) = 4
ioctl(3, FIGETBSZ, 0xbffff8d4)          = 0
brk(0x80c2000)                          = 0x80c2000
close(3)                                = 0
close(4)                                = 0
write(2, "testfile does not have slack\n", 29testfile does not have slack
) = 29
_exit(1)                                = ?
[root@localhost evidence]#
```

Figure 9 – execution of unknown binary using a testfile

As shown earlier in Figure 7, I gained some useful information in a long string of text from the unknown binary. This should help me perform a search to potentially identify the code.

The string was:

"*use block-list knowledge to perform special operations on files*"

This can also be seen when I use the ***strings*** command to search for keywords within the file as well as other key words such as *'newt'* and the date '*07/15/03*':

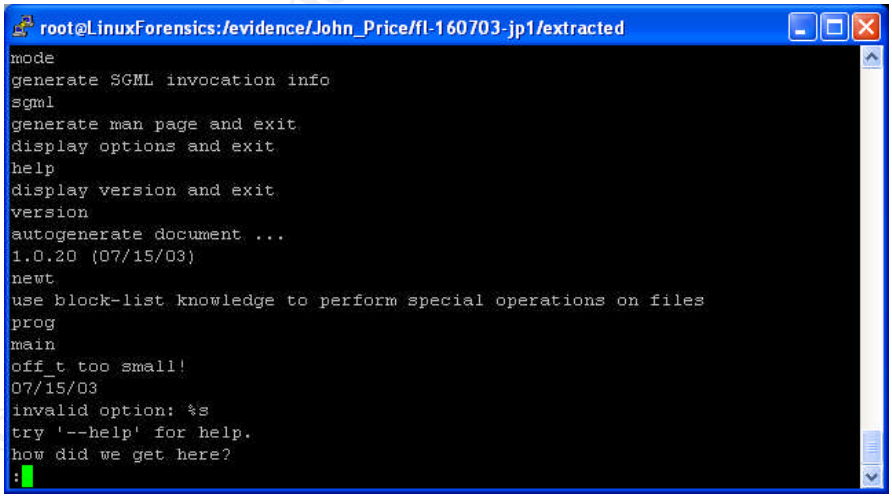


Figure 10 – Sample keywords output using the strings command

This string of text is very specific and could be specific to this program. I use this to search for the file on Google. Amazingly, only two hits, and they are about the same program.

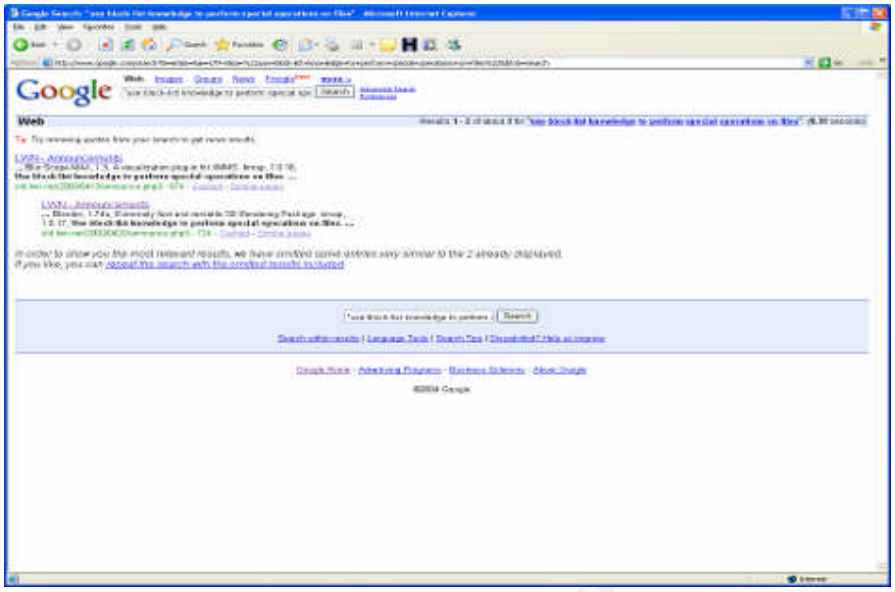


Figure 11 – Locating the real name of the binary using Google

This shows that the selected text appears to be from a program called bmap. I have already identified from the generated help output a suggested version of 1.0.20.

Again a quick Googling, and I find that bmap 1.0.20 source is available from here:

http://garchive.movealong.org/bmap-1.0.20/

To prove this program is bmap 1.0.20, I need to compile the source code and compare the md5 checksums for the resulting binary file with the md5 checksum of the file in evidence.

A quick check on the source will allow me to determine if I am on the right track. I compile the source for bmap-1.0.20 and execute the resulting binary asking for the help page again to compare it against the help page displayed from the unknown binary.

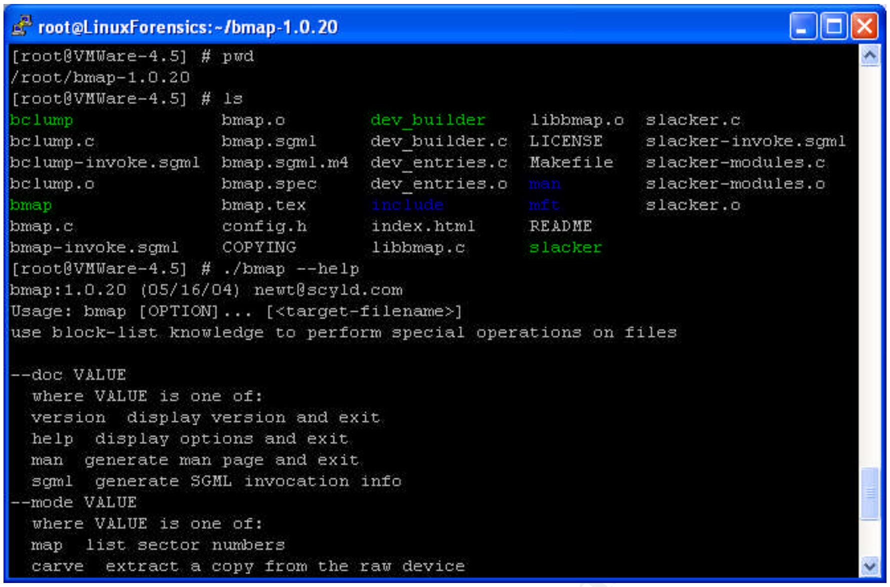


Figure 12 – Quick and dirty verification of binary

As shown above, the output is very similar, but not exact. I now have a problem; the output text differs to that shown in figure 7. They both report that they are version 1.0.20, however the date displayed is different and suspect file has the word "newt" and the other the e-mail address newt@scyld.com. In addition to this, the values output for the –mode option are different as is the help text. I assume that these changes are to obscure the identification of the program should it be discovered.

I may have, however, identified when the unknown binary was compiled. The version I have compiled has the date when it was compiled shown (i.e. 05/16/04). It is likely that the unknown binary was therefore compiled on 15th July 2003.

From the output of autopsy File Analysis feature "Strings Report" or by using the *strings –a* command, I can also find the version of the operating system, and the version of the GCC compiler used to generate the ***prog*** binary:

```
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)
```

Figure 13 – Identification of system that generated the unknown binary

So, I have identified that I need to compile the source I have identified on a Redhat 7.3 system, using GCC 2.96-113 and have the system date set to 07/15/03. In addition to this, I firstly need to edit the source code to make the changes identified earlier.

As shown below, the changes to the code are only made in one place:

```
{"mode","operation to perform on files",
                MOT_VENUM|MOF_SILENT,
                MO_VENUM_CAST{
                        {"map","list sector numbers",
                                0,MO_INT_CAST(BMAP_MAP)},
                        {"carve","extract a copy from the raw device",
                                0,MO_INT_CAST(BMAP_CARVE)},
                        {"slack","display data in slack space",
                                0,MO_INT_CAST(BMAP_SLACK)},
                        {"putslack","place data into slack",
                                0,MO_INT_CAST(BMAP_PUTSLACK)},
                        {"wipeslack","wipe slack",
                                0,MO_INT_CAST(BMAP_WIPESLACK)},
                        {"checkslack","test for slack (returns 0 if file has slack)",
                                0,MO_INT_CAST(BMAP_CHECKSLACK)},
                        {"slackbytes","print number of slack bytes
available",0,MO_INT_CAST(BMAP_SLACKBYTES)},
                        {"wipe","wipe the file from the raw
device",0,MO_INT_CAST(BMAP_WIPE)},
                        {"frag","display fragmentation information for the
file",0,MO_INT_CAST(BMAP_FRAGMENT)},
                        {"checkfrag","test for fragmentation (returns 0 if file is
fragmented)",0,MO_INT_CAST(BMAP_CHECKFRAG)},
                        {NULL,NULL,0,MO_CAST(NULL)}}
                },
        {"outfile","write output to ...",MOT_FILENAME,{NULL}},
        {"label","useless bogus option",MOT_FLAG,{NULL}},
        {"name","useless bogus option",MOT_FLAG,{NULL}},
        {"verbose","be verbose",MOT_FLAG,{NULL}},
        {"log-thresh","logging threshold ...",
```

Figure 14 – Identifying the code to change in the bmap-1.0.20 source

In my Virtual PC environment, Redhat 7.3 is loaded using known sourced ISO images from ftp.redhat.com. I check that the version of the compiler is correct:

```
# gcc –v
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/2.96/specs
gcc version 2.96 20000731 (Red Hat Linux 7.3 2.96-110)
```

I can see from this that the default ISO image of Redhat 7.3 does not appear to have the correct version of the GCC packages installed as it reports version 2.96-110. I do however, compile the code, strip the resulting binary, and compare the MD5 checksums for the resulting binary. The results are indeed different.

By upgrading the gcc package set from 2.96-110 to 2.96-113 I should take one step closer. I upgrade the following rpm's to upgrade to gcc 2.96-113 as identified in the RedHat errata found here: https://rhn.redhat.com/errata/RHBA-2002-200.html

- cpp-2.96-113.i386.rpm
- gcc-2.96-113.i386.rpm
- gcc-c++-2.96-113.i386.rpm

- gcc-chill-2.96-113.i386.rpm
- gcc-g77-2.96-113.i386.rpm
- gcc-java-2.96-113.i386.rpm
- gcc-objc-2.96-113.i386.rpm
- libstdc++-2.96-113.i386.rpm
- libstdc++-devel-2.96-113.i386.rpm

However the resulting binary once compiled, stripped and the md5 checksum produced, does not match the required binary. Using the *strings –a* command to produce the compile strings as before shows an inconsistency.

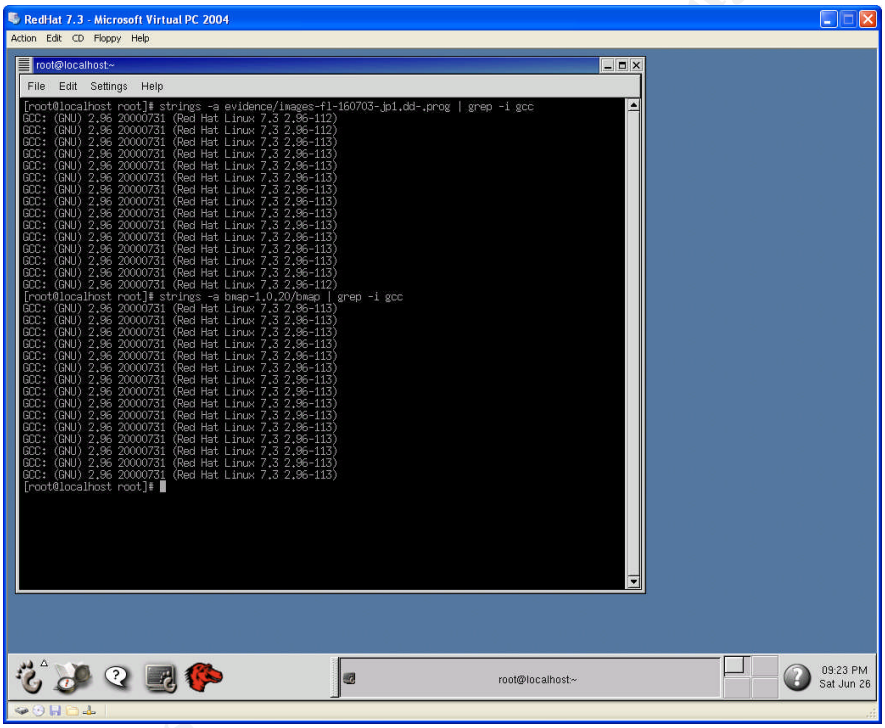


Figure 15 – Comparison of the strings output of the two binaries

It is evident from this that a package is still not at the correct revision. By upgrading the glibc packages to the latest available at *updates.redhat.com* (which at time of writing is 2.2.5-44), I take an interesting step forward as **all** the compiler strings output become GCC 2.96-113. So I have identified that it is indeed the glibc package set that needs upgrading, but I have taken a step too far.

A Redhat archive sites at *redhat.lsu.edu* has the following packages available as superseded versions of glibc:

```
glibc-2.2.5-36.i386.rpm 21-Jun-2002 20:55   3.0M
glibc-2.2.5-37.i386.rpm 15-Jul-2002 21:59   3.0M
glibc-2.2.5-39.i386.rpm 12-Aug-2002 08:13   3.0M
glibc-2.2.5-40.i386.rpm 03-Oct-2002 04:33   3.0M
glibc-2.2.5-42.i386.rpm 05-Nov-2002 23:51   3.0M
```

Figure 16 – List of alternative glibc packages

To identify which of these will give the correct combination of packages, I will have to try them one by one starting from version 36 and ending at version 42 with the required dependent packages.

Gcc-2.96 Version	Glibc-2.2.5 Version	Md5sum
113	2.2.5-34	670a95b577f60dec2cce3be99a5b845d
113	2.2.5-36	7b9248c4fc6a1f704c8f400f255b21c4
113	2.2.5-37	7b9248c4fc6a1f704c8f400f255b21c4
113	2.2.5-39	d7e18d804b8b182c2b9a3394fd4ce377
113	2.2.5-40	d7e18d804b8b182c2b9a3394fd4ce377
113	2.2.5-42	7b80d9aff486c6aa6aa3efa63cc56880
113	2.2.5-44	39c587a38269b48ee3282d40edcc8249
Unknown	unknown	7b80d9aff486c6aa6aa3efa63cc56880

Figure 17 – List of tried compilations, and the resulting md5 output

Note of interest, the md5sum for 2.2.5-36 and 2.2.5-37 were the same, and for 2.2.5-39 and 2.2.5-40.

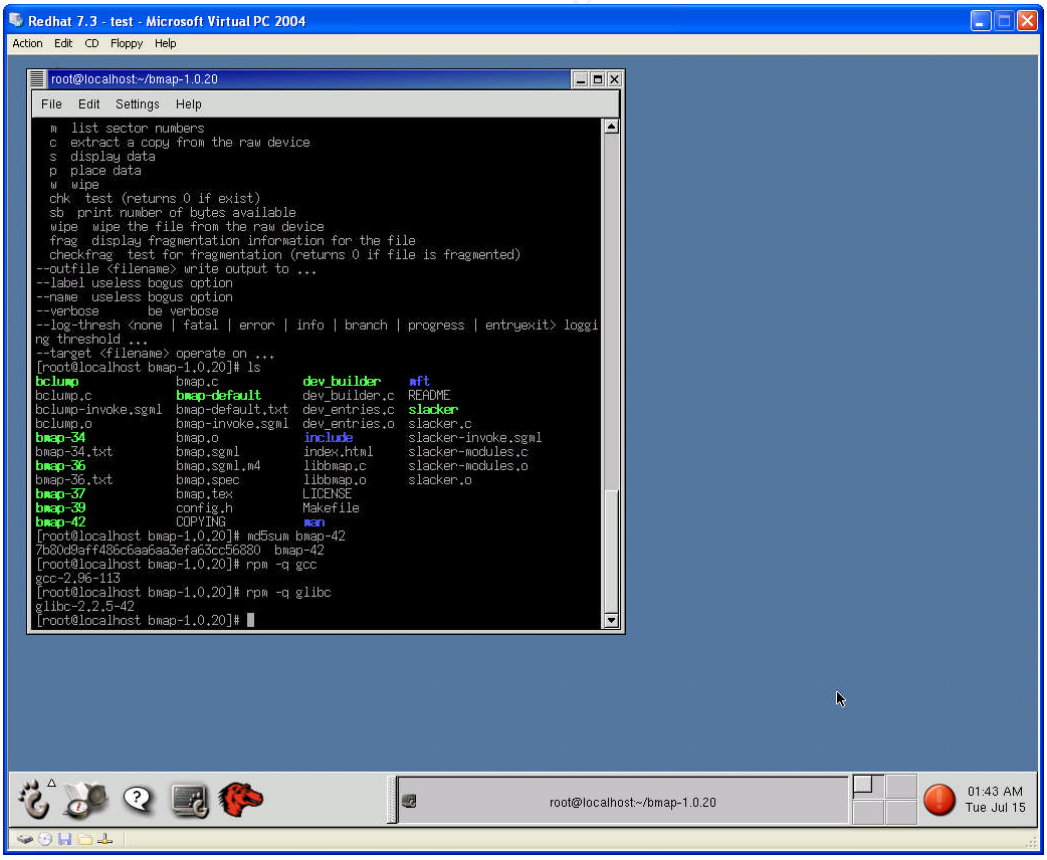


Figure 18 – Identification of the unknown binary

I have managed to identify the unknown binary – prog. As highlighted in the table above, the binary has been identified as:

- Bmap-1.0.20, source code amended to obfuscate the commands offered
- Compiled on a Redhat Linux 7.3 system
- Using GCC 2.9-113 and glibc-2.2.5-42

To discover when the file was created, modified and accessed, I need to produce a MAC timeline. I perform this step within Autopsy using the "File Activity Time Lines" option. From this using the UNIX command grep to parse the file, the following timeline for the prog binary is found:

```
# more body-timeline | grep prog
Mon Jul 14 2003 15:24:00   487476 m.. -/-rwxr-xr-x 502      502       18       /prog
Wed Jul 16 2003 07:05:33   487476 ..c -/-rwxr-xr-x 502      502       18       /prog
Wed Jul 16 2003 07:12:45   487476 .a. -/-rwxr-xr-x 502      502       18       /prog
```

I can see from the above output that the user and group fields are both set to 502. I do not know what these UID and GID entries should be named as I do not have the /etc/passwd and /etc/group files from the system that created the floppy disk. However, if a record exists of the user ID's and group ID's created on John Prices PC, then these may tally with the output giving some indication of a possible link.

Again, from the MAC time analysis output, I have determined that the prog file is 487476 bytes in length. This also tallies with the size of the binary generated from the bmap 1.0.20 source code.

To compare the MD5 checksum with that of the value entered on the evidence log; both an md5 checksum and a sha1 checksum are shown.

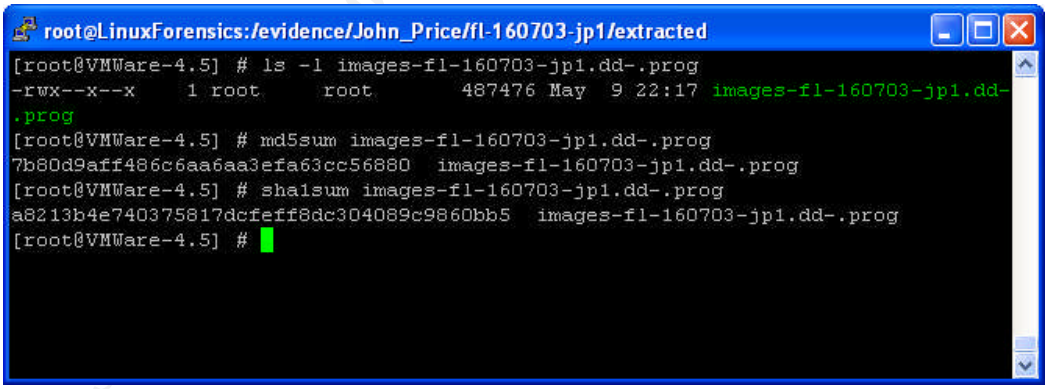


Figure 19 – MD5 hash of the recovered binary

By using the command:

```
# strings -10 images-fl-160703-jp1.dd-.prog
```

The following string output was generated, the output has been trimmed to remove non-ascii text, and repeat lines:

```
mft_getopt                                invalid index %d
argv[%d] is NULL                          argv[%d] (%s) is not an option
```

```
examining a filename or url!
checking against %s
examining an enum!
examining a venum!
arg matches against %s
matches against %s
mft_log_init
MFT_LOG_THRESH
unspecified

<table
bgcolor=%s><tr><td>%s</td></tr></table><br>
.TH %s "%d" "%s" "%s" "%s"
Report bugs to %s.
 [<%s-filename>]
--%s <int> %s
--%s VALUE
<tt>%s</tt> invocation
 [&lt;%s-filename&gt;]

<tag>--%s</tag>    %s
<tag>--%s &lt;int&gt;</tag> %s
<tag>--%s &lt;
<tag>--%s VALUE</tag>
</descrip>
operate on ...
log-thresh
useless bogus option
test for fragmentation (returns 0 if file
is fragmented)
wipe the file from the raw device
test (returns 0 if exist)
display data
list sector numbers
generate SGML invocation info
display options and exit
autogenerate document ...
use block-list knowledge to perform special
operations on files
invalid option: %s
how did we get here?
target filename: %s
%s is not a regular file.
Unable to open file: %s
target file block size: %d
Unable to determine count
%s has holes in excess of %ld bytes...
nul block while mapping block %d.
read error
%s fragmented between %d and %d
file size was: %ld
block size: %d
# File: %s  Location: %Ld  size: %d
%s has slack
%s has fragmentation
bmap_get_slack_block
Unable to stat fd
error getting block count
mapping block %lu

error mapping block %d. block returned 0
unable to stat fd
filesystem reports 0 blocksize
stat reports %d blocks: %d
bmap_map_block
bmap_raw_open
Unable to stat file: %s
unable to determine raw device of %s
device mismatch 0x%x != 0x%x
raw fd is %d
/.../image
Wrong medium type
Disk quota exceeded
Is a named type file
Not a XENIX named type file
Stale NFS file handle
Operation already in progress
```

```
%s is a well-formed argument
flagized option invokation
matched against an enum val
matched against an venum val
process_match
invalid value for enum
nbd-server
mft_log_shutdown
<table bgcolor=%s><tr><td>%s:
%s</td></tr></table><br>
<table
bgcolor=%s><tr><td></td></tr></table><br>
.SH SYNOPSIS
Usage: %s [OPTION]...
--%s <arg> %s
--%s <filename> %s
  where VALUE is one of:
<tt>%s [&lt;OPTIONS&gt;]
Where <bf>OPTIONS</bf> may include any
of:
<tag>--%s &lt;arg&gt;</tag> %s
<tag>--%s &lt;filename&gt;</tag> %s
&gt;</tag> %s
<tag>%s</tag>  %s
<tag>--%s</tag> %s
logging threshold ...
be verbose
write output to ...
display fragmentation information for the
file
print number of bytes available
place data
extract a copy from the raw device
operation to perform on files
generate man page and exit
display version and exit
1.0.20 (07/15/03)
off_t too small!

try '--help' for help.
no filename. try '--help' for help.
Unable to stat file: %s
%s has multiple links.
Unable to determine blocksize
unable to raw open %s
Unable to allocate buffer
error mapping block %d (%s)
seek failure
write error
getting from block %d
slack size: %d
seek error
stuffing block %d
%s does not have slack
%s does not have fragmentation
NULL value for slack_block
Unable to determine blocksize
fd has no blocks
error mapping block %d. ioctl failed with
%s
bmap_get_block_count
unable to determine filesystem blocksize
computed block count: %d
bmap_get_block_size
nul block while mapping block %d.
NULL filename supplied
%s is not a regular file.
unable to stat raw device %s
unable to open raw device %s
bmap_raw_close
write error
No medium found
Remote I/O error
No XENIX semaphores available
Structure needs cleaning
Operation now in progress
No route to host
```

```
Host is down                          Connection refused
Connection timed out                  No buffer space available
Connection reset by peer              Network is unreachable
Network is down                       Address already in use
Protocol family not supported         Operation not supported
Socket type not supported             Protocol not supported
Protocol not available                Message too long
Destination address required          Too many users
Streams pipe error                    Remote address changed
File descriptor in bad state          Name not unique on network
Bad message                           RFS specific error
Multihop attempted                    Protocol error
Communication error on send           Srmount error
Advertise error                       Link has been severed
Object is remote                      Package not installed
Machine is not on the network         Out of streams resources
Timer expired                         No data available
Device not a stream                   Bad font file format
Invalid slot                          Invalid request code
Exchange full                         Invalid request descriptor
Invalid exchange                      Level 2 halted
No CSI structure available            Protocol driver not attached
Link number out of range              Level 3 reset
Level 3 halted                        Level 2 not synchronized
Channel number out of range           Identifier removed
No message of desired type            Directory not empty
Function not implemented              No locks available
File name too long                    Resource deadlock avoided
Numerical result out of range         Broken pipe
Too many links                        Read-only file system
Illegal seek                          No space left on device
File too large                        Text file busy
Too many open files                   Too many open files in system
Invalid argument                      Is a directory
Not a directory                       No such device
Invalid cross-device link             File exists
Device or resource busy               Block device required
Bad address                           Permission denied
Cannot allocate memory                No child processes
Bad file descriptor                   Exec format error
Argument list too long                No such device or address
Input/output error                    Interrupted system call
No such process                       No such file or directory
Operation not permitted               Too many references: cannot splice
Cannot send after transport endpoint  Transport endpoint is not connected
shutdown
Transport endpoint is already connected  Software caused connection abort
Network dropped connection on reset   Cannot assign requested address
Address family not supported by protocol  Protocol wrong type for socket
Socket operation on non-socket        Interrupted system call should be
                                      restarted
Invalid or incomplete multibyte or wide  Cannot exec a shared library directly
character
Attempting to link in too many shared  .lib section in a.out corrupted
libraries
Accessing a corrupted shared library  Can not access a needed shared library
Value too large for defined data type  Too many levels of symbolic links
Numerical argument out of domain      Inappropriate ioctl for device
Resource temporarily unavailable      TRIM_THRESHOLD_
MMAP_THRESHOLD_                       system bytes     = %10u
in use bytes     = %10u               Total (incl. mmap):
max mmap regions = %10u               max mmap bytes   = %10lu
malloc: top chunk is corrupt          free(): invalid pointer %p!
malloc: using debugging hooks         realloc(): invalid pointer %p!
Unknown error                         ANSI_X3.4-1968//TRANSLIT
syslog: unknown facility/priority: %x  out of memory [
```

Use of bmap

What type of program is it?

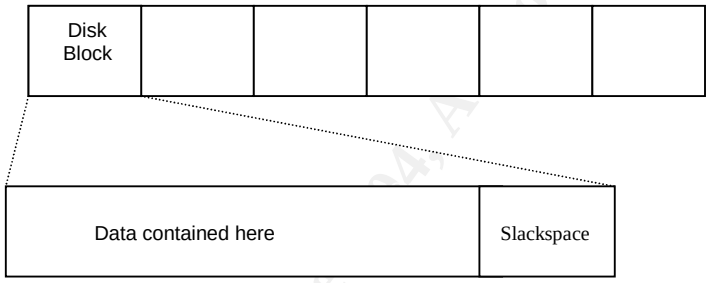
As I discussed earlier, bmap:

- Is in ELF format
- Is for a 32 bit, little Endian platform
- Is compiled for an Intel platform
- Is statically linked
- Is stripped of any symbol table
- Originated from a Linux 2.2.5 system
- Was compiled on a Redhat 7.3 system
- Compiled using GCC 2.96-113
- Compiled against glibc-2.2.5-42

What is it used for?

bmap is an Linux utility to add, recover, and check for data contained within "slack space" found in file systems. The utility also has the ability to report if a file as the ability to store information in slack space.

Slack space is the space that is often found between the end of a file, and the end of the block that the file is stored in. This space can have many uses; on Windows systems it can contain data that would otherwise be overwritten, on Linux systems the slack space is normally zeroed out to the end of the block.



The normal filesystem analysis utilises such as Virus Scanners may not analyse the content of the slackspace as they are only interested in the contents of the file, not the underlying disk structure holding the file.

When was the last time it was used?

From the MAC timeline shown earlier, I have the entry:

```
Wed Jul 16 2003 07:12:45   487476 .a. -/-rwxr-xr-x 502      502      18       /prog
```

This shows us the last time the prog executable was accessed. This is often the equivalent of the last time the command was executed. Therefore I can state that the file ***prog*** was last accessed from the floppy disk at 07:12:45 on July 16th 2003 and that this could be the time it was executed.

Step-by-step analysis

If I use the captured version of bmap, I can check each of the files on the captured image to see if it contains slack space.

I mount the iso image:

```
# mount –o loop,ro /mnt/iso /evidence/John_Price/fl-160703-jp1/images/fl-160703-jp1.dd
```

The options are:

- loop – mount a raw disk image as a file system
- ro – mount the file system in read only mode

Further options exist that I could use, but for completeness:

- nodev – do not allow devices
- noexec – do not allow binaries to be executed
- noatime – do not allow changes of inode atime

By analysing each of the files on the image in turn to see if bmap reports that the file contains slack space, I discover that only one file reports that it has slack as can be seen below:

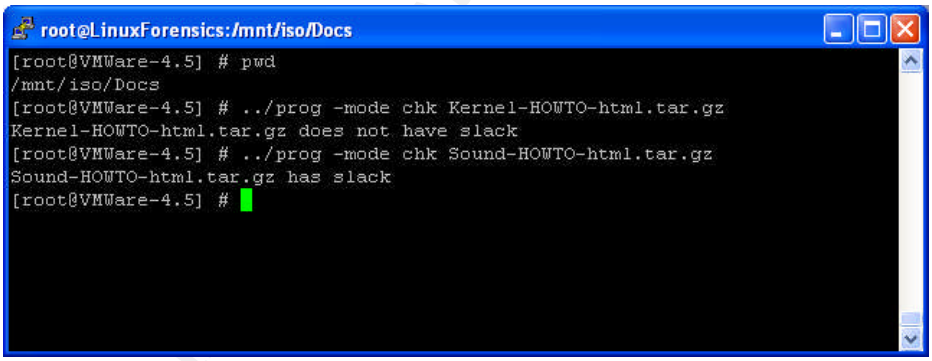


Figure 20 – Aha! Slack space has been discovered

Recovery of slack space data

Again I can use bmap to recover the data that has been put into slack space. Rather than use the recovered version of the command, I use the compiled version as I have proved that it is the same program as the unknown prog file.

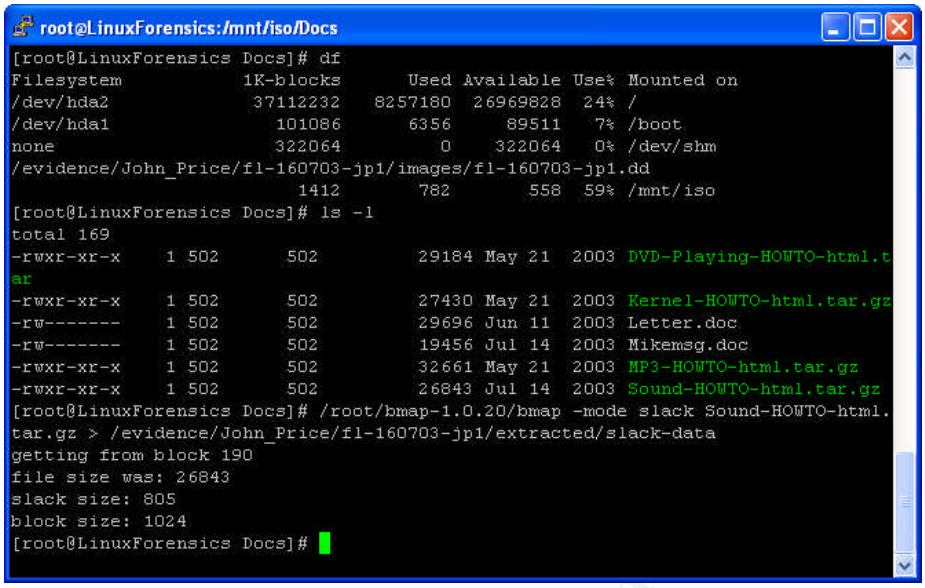


Figure 21 – recovery of the slack space hidden data

The bmap command is used as follows:

```
# bmap –mode slack Sound-HOWTO-html.tar.gz > /evidence/John_Price/fl-160703-
jp1/extracted/slack-data
```

`-mode` ***`slack`***	outputs the contents of the slack data
`Sound-HOWTO-html.tar.gz`	is the file which I have shown to contain slack data
`slack-data`	contains the extracted slack data file.

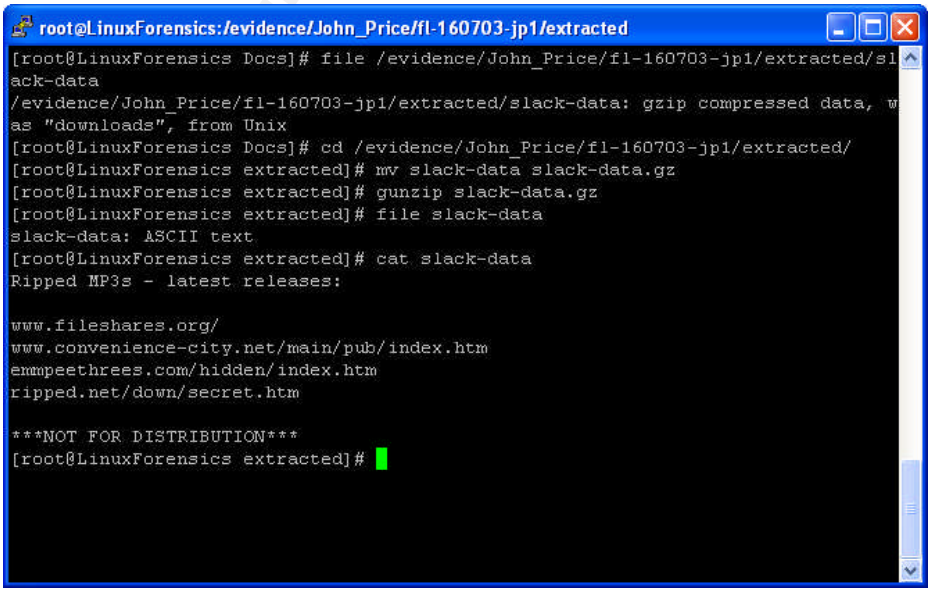


Figure 22 – The smoking gun?

I examine the outputted file slack-data and by using the file command I discover that it was originally a gzip compressed data file created from the file named “downloads”. This file was also initially created on a UNIX system.

To extract the contents, I rename the file so that gzip finds the expected .gz extension, and gunzip the file. The contents of the file are shown above in figure 23.

The following information was obtained for these web sites:

Fileshares.org, convenience-city.net, and emmpeethrees.com no longer exist as registered domain names.

However, ripped.net is still online:

```
Softline Studios
   PO Box 2004
   Del Mar, CA 92014
   US

   Domain Name: RIPPED.NET

   Administrative Contact::
        Loren Stocker: loren@800.net
        Softline Studios
        PO Box 2004
        Del Mar, CA 92014
        US
        Phone:: +1.858.792.5000
        Fax::
   Technical Contact::
        Loren Stocker: loren@800.net
        Softline Studios
        PO Box 2004
        Del Mar, CA 92014
        US
        Phone:: +1.858.792.5000
        Fax::

   Record updated date on: 2003-06-29 18:36:07
   Record created date on: 2001-06-30
   Record will be expiring on date: 2004-06-30
   Database last updated on: 2004-06-29 09:05:39 EST

   Domain servers in listed order:

   NS1.STEALTHREGISTRY.COM       64.175.161.93
   NS2.STEALTHREGISTRY.COM       64.175.161.94
```

Figure 23 – Registry information for ripped.net

As shown in Figure 24, the domain name servers which are registered as the primary name servers are owned by stealthregistry.com

The domain ripped.net only exists within NS1, and NS2 returns an error when the query is made.

Using the NS1 of stealthregistry.com, I find that the domain ripped.net is hosted on an ADSL line at the IP address 64.175.161.93. This is also the IP address of the DNS server, so the information may be incorrect due to a miss configuration or the site is hosted on the DNS server potentially without the knowledge of stealthregistry.com

To give some indication of the possible location of the ADSL line, I use the *traceroute* command to see what DNS information exists.

```
[root@LinuxForensics root]# traceroute 64.175.161.93
traceroute to 64.175.161.93 (64.175.161.93), 30 hops max, 38 byte packets
 1  X.X.X.1 (X.X.X.1)  1.520 ms  0.661 ms  0.673 ms
 2  Y.Y.Y.1 (Y.Y.Y.1)  1.517 ms  1.354 ms  1.331 ms
 3  lon1-adsl1.nildram.net (195.149.20.11)  14.678 ms  14.293 ms  15.504 ms
 4  lon1-9.nildram.net (213.208.106.194)  15.381 ms  15.146 ms  24.257 ms
 5  lon1-11.nildram.net (195.149.20.137)  15.260 ms  15.783 ms  15.699 ms
 6  0125.ge-0-0-0.gbr1.ltn.nac.net (64.21.115.61)  17.485 ms  17.141 ms  18.049 ms
 7  0.so-0-3-0.gbr2.nwr.nac.net (209.123.11.209)  84.070 ms  84.583 ms  84.084 ms
 8  3.gig1-2.esd1.nwr.nac.net (209.123.11.190)  183.293 ms  119.552 ms  204.413 ms
 9  ex1-g8-0s1.eqnwnj.sbcglobal.net (206.223.131.79)  83.930 ms  84.704 ms  84.630 ms
10  bb2-p13-0.nycmny.sbcglobal.net (151.164.189.146)  84.884 ms  84.457 ms  87.020 ms
11  ex1-p8-0.nycmny.sbcglobal.net (151.164.240.222)  85.659 ms  85.425 ms  85.090 ms
12  core2-p3-0.crhnva.sbcglobal.net (151.164.188.198)  91.147 ms  91.424 ms  91.265 ms
13  core2-p3-0.cratga.sbcglobal.net (151.164.241.94)  101.762 ms  102.445 ms  101.428
ms
14  core1-p1-0.cratga.sbcglobal.net (151.164.241.81)  123.885 ms  277.381 ms  211.278
ms
15  core2-p11-0.crhstx.sbcglobal.net (151.164.240.113)  114.321 ms  114.989 ms
115.893 ms
16  core2-p3-0.crrvca.sbcglobal.net (151.164.241.126)  150.753 ms  150.700 ms  150.612
ms
17  bb2-p14-3.irvnca.pbi.net (151.164.241.118)  151.288 ms  152.993 ms  151.604 ms
18  bb1-p3-0.irvnca.sbcglobal.net (151.164.191.205)  152.916 ms  151.449 ms  152.369
ms
19  bb1-p8-0.sndg02.sbcglobal.net (151.164.188.110)  153.404 ms  153.029 ms  152.959
ms
20  dist1-vlan40.sndg02.pbi.net (63.200.206.4)  153.173 ms  153.423 ms  152.855 ms
21  rback10-fe2-0.sndg02.pbi.net (63.200.206.141)  152.688 ms  153.279 ms  154.083 ms
22  adsl-64-172-235-30.dsl.sndg02.pacbell.net (64.172.235.30)  164.273 ms  163.202 ms
161.747 ms
23  adsl-64-175-161-93.dsl.sndg02.pacbell.net (64.175.161.93)  166.588 ms  165.209 ms
165.459 ms
```

Figure 24 – Traceroute details to host which is hosting ripped.net

The host adsl-64-175-161-93.dsl.sndg02.pacbell.net is either hosting ripped.net, or has been compromised and is unwittingly hosting the site.

However, quick surf of the IP address for ripped.net shows that it is hosting a holiday site:

Figure 25 – Is ripped.net a hosted, hacked or hidden site?

However, the IP address 64.175.161.93 is hosting a redirect to www.yahoo.com

```
[root@LinuxForensics root]# telnet 64.175.161.93 80
Trying 64.175.161.93...
Connected to adsl-64-175-161-93.dsl.sndg02.pacbell.net (64.175.161.93).
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 302 Found
Date: Tue, 29 Jun 2004 12:04:48 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Location: http://www.yahoo.com/
Content-Length: 285
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.yahoo.com/">here</a>.</p>
<hr />
<address>Apache/2.0.40 Server at pics.carmelvalley.net Port 80</address>
</body></html>
Connection closed by foreign host.
```

Figure 26 – Virtual hosting confirmed

It would appear therefore, that the site is hosting a number of virtual web sites, depending on the URL supplied. The address string of the host "pics.carmelvalley.net" may give us some information to follow. If I perform a DNS lookup on the domain name, I get the same IP address as ripped.net

```
[root@LinuxForensics root]# nslookup pics.carmelvalley.net
Note:  nslookup is deprecated and may be removed from future releases.
Consider using the `dig' or `host' programs instead.  Run nslookup with
the `-sil[ent]' option to prevent this message from appearing.
Server:         195.112.4.4
Address:        195.112.4.4#53

Non-authoritative answer:
Name:   pics.carmelvalley.net
Address: 64.175.161.93
```

Figure 27 – more investigation from the output from ripped.net

By again using Google, and also surfing to www.carmelvalley.net, I find that the contact for the domain is someone called Loren, as the only displayed content on the site is:

Call Loren @ (858) 792-5000 for details.

By using this telephone information I find the following information:

Exchange	**Exchange Location:** DEL MAR

Again, a quick Google returns some more information. Loren Stocker is the managing director of this hosting site. If it becomes clear from further analysis that ripped.net should be a hacked site then Loren Stocker should be informed.

Is there anything else I can find?

The hidden file was entitled “downloads”. I am able to search for any further reference of this keyword by performing a keyword search of the disk image.

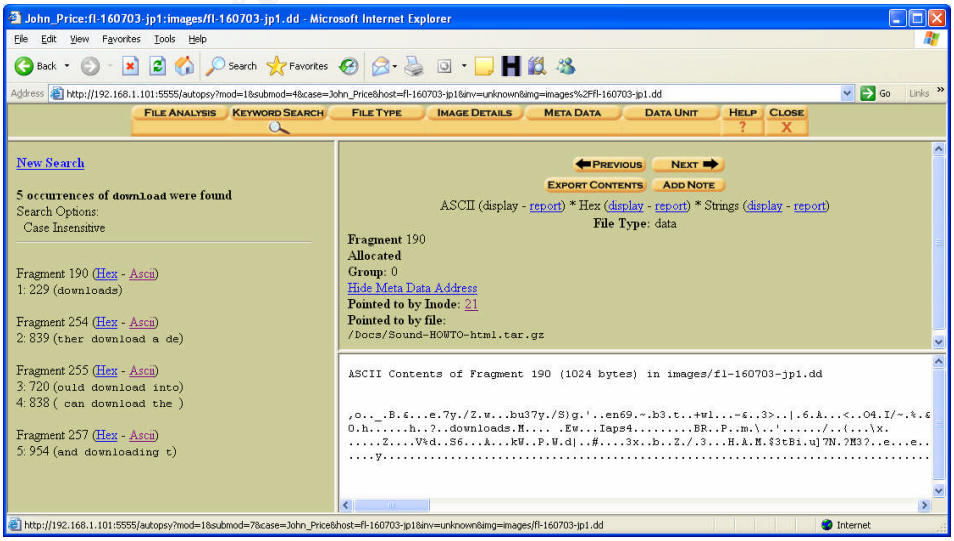


Figure 28 – Keyword search for “downloads”

The keyword is located four times, with the only suspect hit being within the recovered slackspace binary. The keyword search was repeated for the unallocated disk space, and no hits were found.

As this is a small filesystem, I can contemplate running a Lazarus analysis of the system. Lazarus attempts to identify disk blocks within an image. Unfortunately, Lazarus can take a considerable amount of time to run on larger file systems. Interestingly, I scan the disk image, and Lazarus detects some sections of disk blocks that it believes are sound files, as shown below:

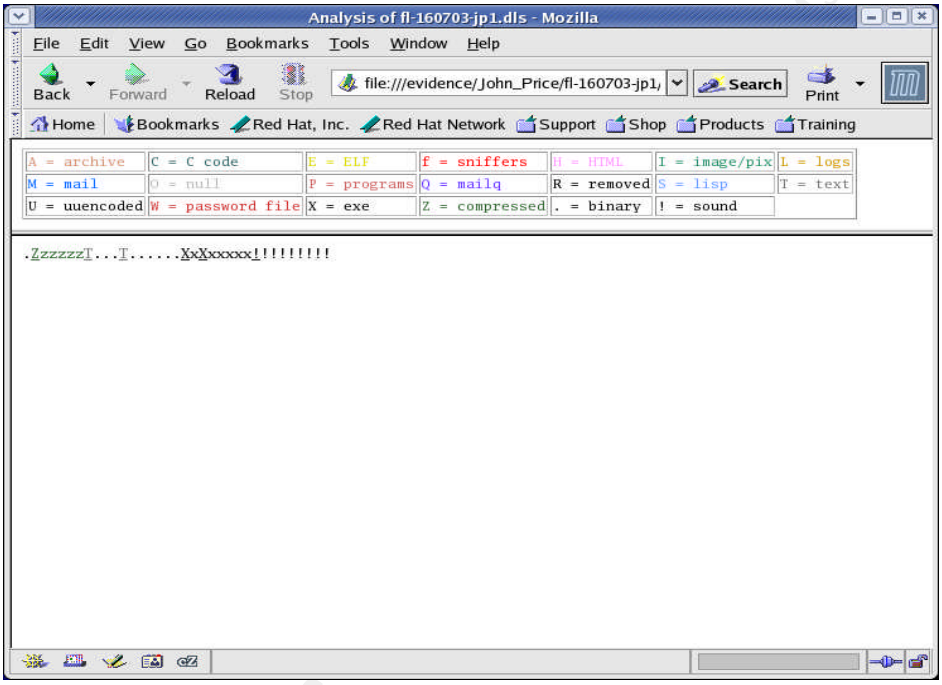


Figure 29 – Lazarus output indicating sound file

The block referenced by the first "!" character is block 132. There are two methods of resolving this block number to the complete forensic image.

Firstly, I can use the command dcalc to convert the block number:

Figure 30 - using dcalc to convert the Lazarus block number

The number output is referenced from 1, whereas block numbers start from 0, so I must look from block number 938 onwards. I can then use this number to export the blocks from the image using the dcat command and calculating the number of blocks needed.

The second method is again to use the autopsy forensic browser to do this work for me, as shown below:

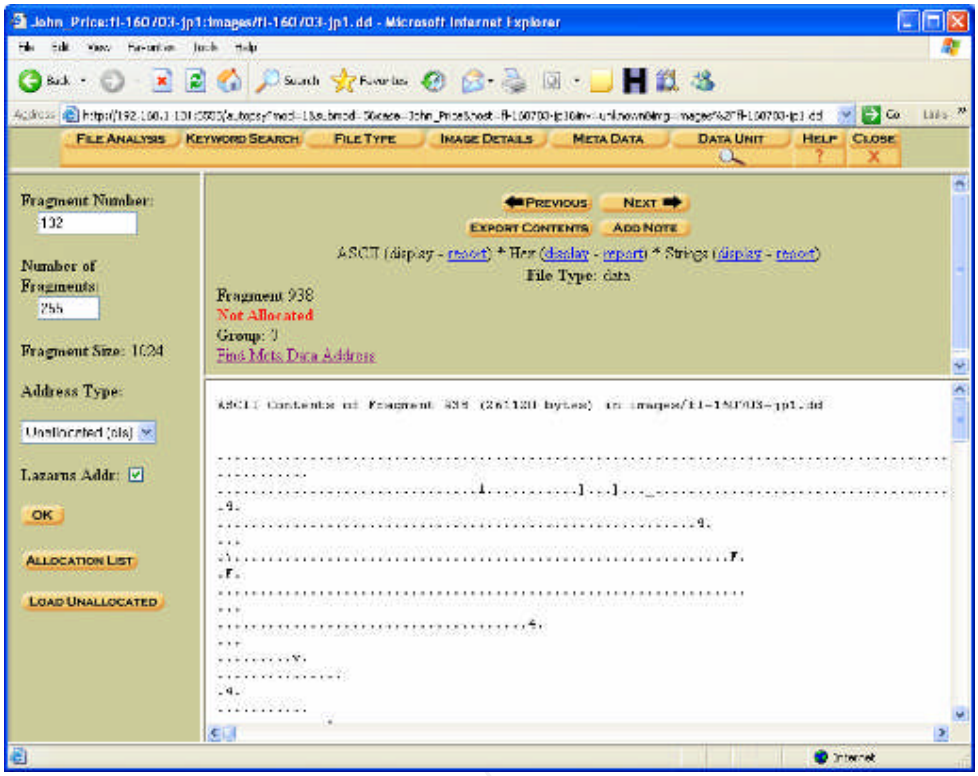


Figure 31 – using autopsy to decode Lazarus block numbers

By clicking on “Export Contents” I was able to dump the file to my local PC, the resulting file was named “autopsy.mp3” however Microsoft Windows Media player 10 refused to play the file.

Lazarus can be wrong in determining the file type, so I need to detect if the sniff of information here is more than just a sniff. I download a tool that can detect MP3 files within a piece of data:

Mp3tool, from http://empeg-hijack.sourceforge.net/mp3tool.html is written by Mark Lord, and I must thank Mark for his help in understanding the format of MP3 files.

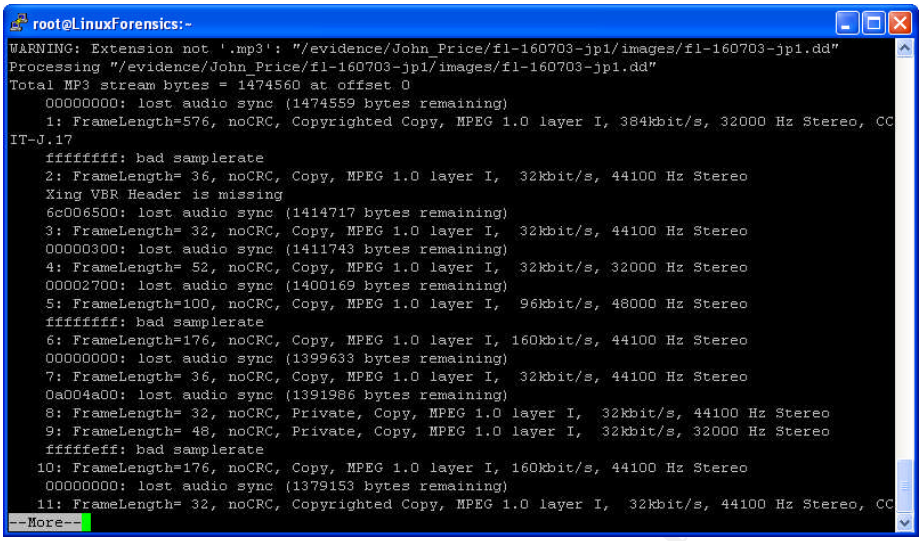


Figure 32 – Scanning the disk image for potential MP3 data

MP3 files have no header, or trailer per se, they have frame information describing the next block of music data; therefore lot of potential MP3's are detected. But as I can see from the above example, a lot of the MP3 frames are tiny, and are not likely to be real MP3 data but false positives.

An alternative tool to use here is *foremost* to detect the MP3 files, but as foremost requires defined headers and footers to a file, and MP3 has frame headers and footers which can vary frame to frame this would be as inconclusive as using mp3tool.

Although a large number of potential mp3 fragments were found, no playable mp3 information was detected on the floppy disk image.

Other information found on the floppy disk image

Two graphical images were found on the floppy disk as shown below:

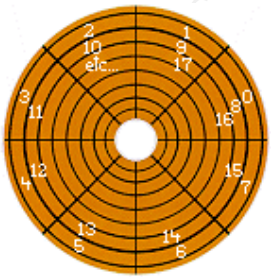


Figure 33 - Recovered .gif image 1

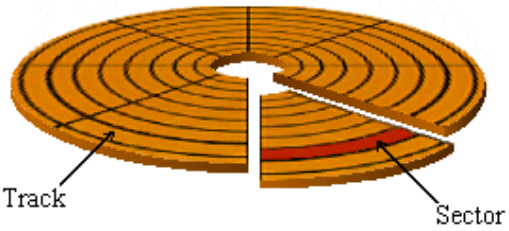


Figure 34 - Recovered .gif image 2

These graphics may have been used to explain the concept of slack space to John Price by the person supplying the custom version of bmap used, or used

by John Price to spread the use of the utility to other colleagues or friends. In addition to this, an apparent image of a screenshot taken from ebay was found, this is shown below:

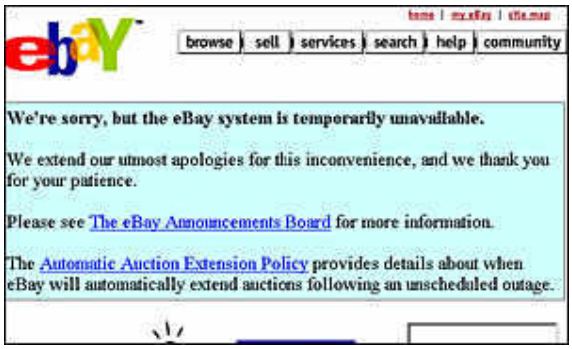


Figure 35 – unknown ebay image

As this image is of poor quality (i.e. it is blurred indicating that the image has been degraded), and a jpeg image, I have used StegDetect[3] to check if any hidden data was stored in the image and thus reducing the image quality. None was found. However, the image could indicate that John Price has an ebay account where he is auctioning MP3 files.

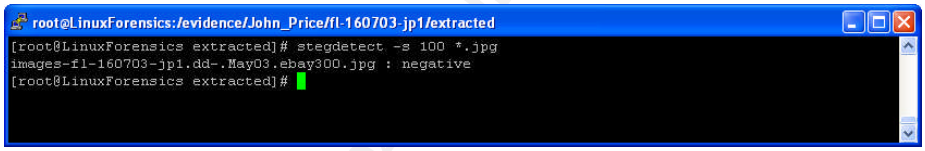


Figure 36 – using stegdetect to check recovered jpeg file

Although not at all conclusive evidence, a search on ebay for a seller named John_Price returns a hit, as does JohnPrice:

[3] http://www.outguess.org/detection.php

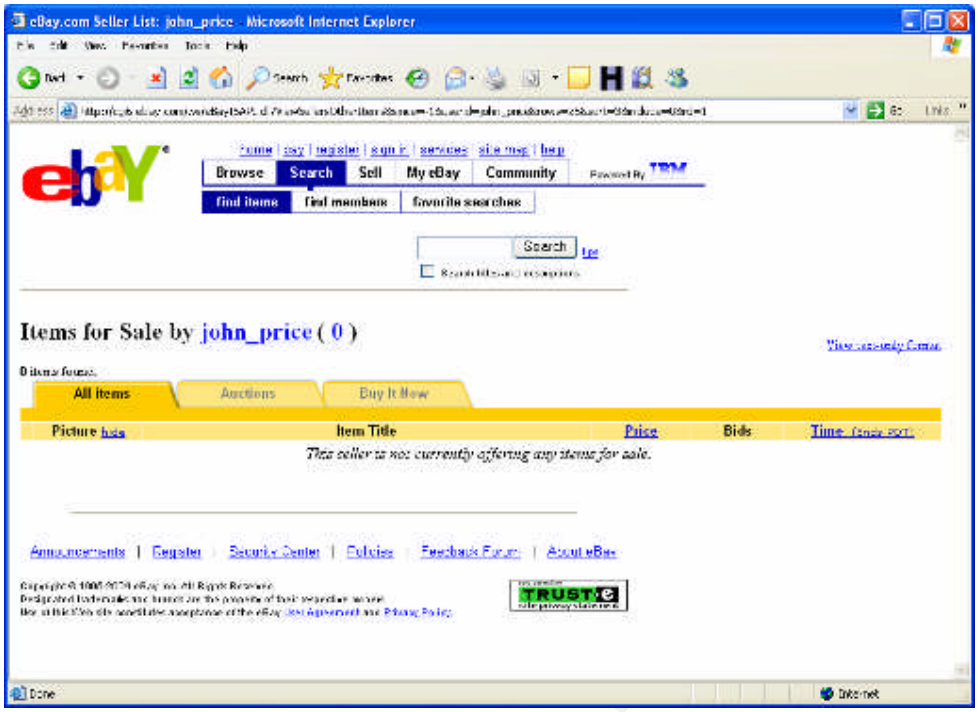


Figure 37 – John Price's ebay account?

Potential Interview Questions

- From the clocking system, I see that you were in early on the 16th July, what were you doing?
- The tool called prog was a cool idea! Who showed you how to use it as this is specialist stuff and way above your skills?
- Who's Mike? And why were you writing a letter to him?
- Has anyone been using your PC before it died?
- Do you buy your own floppies, or does stationary have them as standard – as their a bit old fashioned now?

Further questions to discover more information could be:

- I thought that the quality of most MP3's sucked when I first looked at them, but I found that you could tweak them by using Variable Bit Rate, rather than constant. Which do you use when ripping?
- We spoke to Loren in California, some interesting web sites that he runs
- I've been considering selling some old PC kit on ebay, but I've never used it. What sort of username would you use? Do you have one?

Case Information

Other than the content of the download file, I found no other evidence on the floppy disk. However, I did find four MP3 sites referenced. I could ask for the Systems Administrators to scan the proxy logs of the company Internet

gateway to see if the work ISP links were used to download the files – this would further incriminate the suspect.

A Microsoft Word document was recovered, and the final text is shown below.

```
[root@LinuxForensics extracted]# pwd
/evidence/John_Price/fl-160703-jp1/extracted
[root@LinuxForensics extracted]# /root/bin/antiword -t -s images-fl-160703-jp1.dd-
.Docs.Mikemsg.doc

Hey Mike,

I received the latest batch of files last night and I'm ready to rock-n-
roll (ha-ha).

I have some advance orders for the next run. Call me soon.

JP
```

Figure 38 – Text of the recovered Mikemsg.doc file

This document, as shown in figure 35, was analysed by moving the document to a Windows System. To preserve the integrity of the file, an md5sum was taken before and after the move to allow the files to be compared:

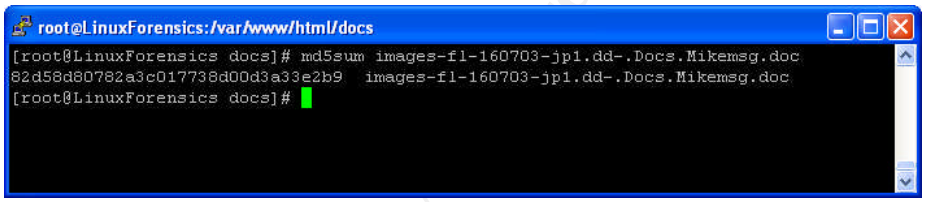


Figure 39 – MD5SUM of the extracted document on the Forensic Workstation

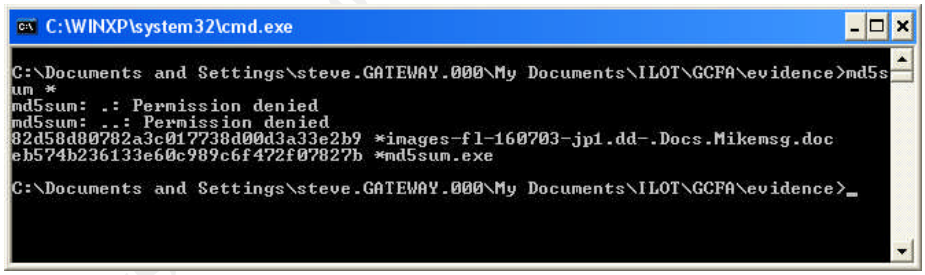


Figure 40 – MD5SUM of the extracted document on the Windows Host

Windows XP shows the properties of the file indicating that the author of the file was John Price, although this is easily changed, and the date the file was created:

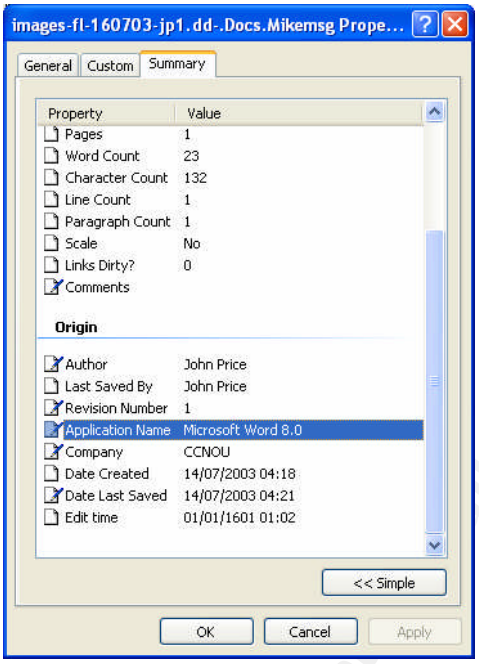


Figure 41 – Output from Windows Properties for the extracted document

As Microsoft Windows has two sets of time stamps, I can also look at the MAC time analysis to see what it shows about the creation of the file. I can see from the figure below that the file appears to have been created at 15:48:15 on Monday, 14th July 2003.

```
[root@LinuxForensics output]# grep -i mike body-timeline
Mon Jul 14 2003 15:48:15    19456 mac -/-rw------- 502      502      17
/Docs/Mikemsg.doc
```

Figure 42 – Verifying the MAC date of the Mikemsg.doc file

By loading the Microsoft Word file into a hex editor, in this case Khexedit, I can also see where the file was actually created:

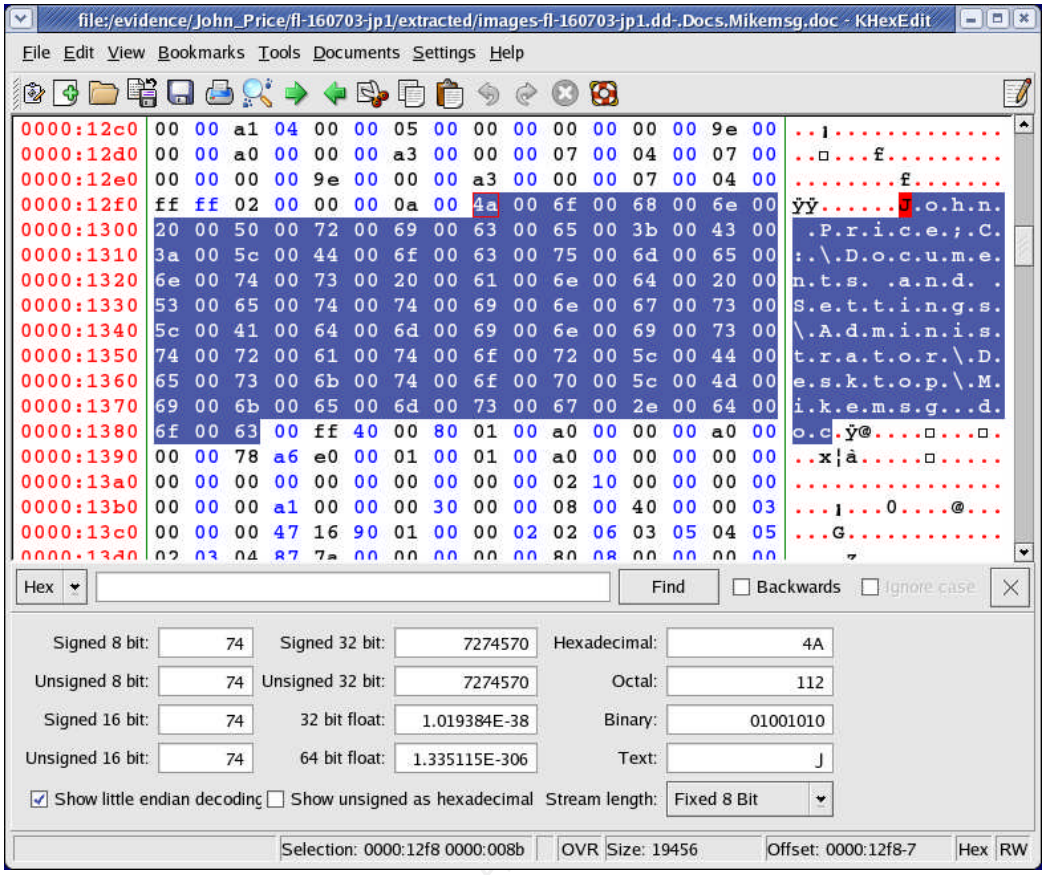


Figure 43 – Hex dump of the extracted Word Document.

I can see from the above picture that the Microsoft Word file extracted from the disk appears to have been previously saved at the location:

C:\Documents and Settings\Administrator\Desktop\Mikemsg.doc

This may be of concern as John Price would appear to have Administrator access on a PC. Even if he should have administration access to his Microsoft Windows PC, he was using a company system to generate a Microsoft Word document which may suggest that he was involved in the illegal distribution of copyright material.

A valid Redhat rpm file was found on the system; this was for NetCat Version 1.10-16 and contained the binary nc, which would be installed in the /usr/bin directory.

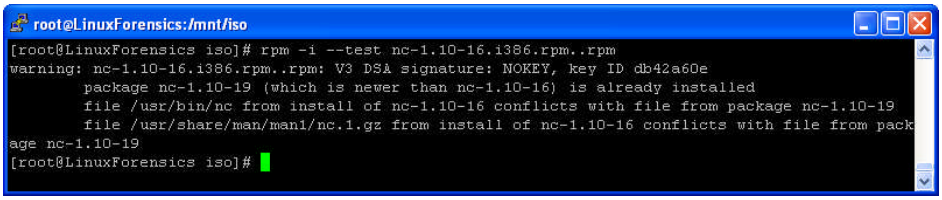


Figure 44 – The netcat package

As this package does not appear to have been installed on the floppy, there is a chance that it may be installed on other company systems. As this utility can be used to tunnel network connections from one system to another, this could allow other systems to be used remotely to store data.

System administrators should check their systems to see if an unauthorised installation of netcat has taken place. It is possible that John Price was using netcat to remotely pipe mp3 files from one system to another, and then using bmap to write this information to that disk.

A quick analysis of the bmap source code shows that the program returns a 1 when the file does not have slack, and returns a 0 when a file does have slack.

```
if(flag_mode!=BMAP_CHECKSLACK && flag_mode!=BMAP_CHECKFRAG)
        {
                retval=0;
        } else if(flag_mode==BMAP_CHECKSLACK) {
                if(retval==0) {
                        mft_logf(MLOG_INFO,"%s has slack",filename);
                } else if(retval==1) {
                        mft_logf(MLOG_INFO,"%s does not have slack",filename);
                }


        return retval;
```

This would allow for systems administrators to use the identified binary bmap to quickly scan computer systems for slackspace, and to recover any further evidence that may still exist.

However, bmap does not allow searches to be made recursively. To get over this, a combination of the bmap command and the find command can be useful:

```
# find /pathtosearch -exec /path/to/bmap/bmap --mode checkslack {} \; 2>&1 | grep 'has
slack'
```

Legal Implications

Although I can assume that the *prog* utility was executed, as the MAC Timeline shows the file being accessed, and a file was found containing illicit data stored in slackspace, this itself is not illegal.

However, the letter between John Price and Mike may constitute an offence under common law in that if intent between John and Mike to profit from the selling of MP3's was can be shown then this is illegal.

Under the ruling of *Scott v Metropolitan Police Commissioner [1975] AC 819*[4], agreeing to an act which may injure a third party's proprietary rights over copyrighted material would be classed as *conspiracy to defraud*.

[4] http://www.fact-uk.org.uk/legislation.htm

Section 12(1) of the Criminal Justice Act 1987 allows for a maximum penalty of 10 years imprisonment and/or an unlimited fine.

Additional Information

Information on slack space, and it use can be found here:

http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html

Information of MP3 file header information can be found here:

http://www.wotsit.org/search.asp?page=5&s=music

For more information on how Copyright theft is handled in the UK, then you should contact FACT at:

http://www.fact-uk.org.uk

For online access to UK acts of parliament, then you go to the source at:

http://www.legislation.hmso.gov.uk

Part 2 - Option 1: Perform Forensic Analysis on a system

Synopsis of Case Facts

I was asked to perform a Forensic Investigation of a computer system which may have been compromised – in that an unauthorised person may have accessed the computer and used it for as yet unknown purposes.

A Forensic Investigation is an investigation to discover what has happened to a computer system. This is performed using techniques and methods that allow me to investigate the computer without disturbing any valuable evidence that may be found.

To obtain a compromised system which was truly in an unknown state I have used a technology referred to as a honeypot. As with a real honey pot, where the honey is seen as being irresistible to the bee a computer honeypot is designed to be irresistible to a computer attacker often referred to as a hacker.

The honeypot system was connected to the Internet. The computer was using a popular computer program called Linux. This program is the heart of the computer system and is referred to as the Operating System. The Operating System controls how a computer functions and what it is able to do. The facilities that the computer can provide are often referred to as services.

A number of such services were made available to any potential hacker. As the version of the Operating System was quite old, many errors have been found in the way it works. These errors, or bugs as they are commonly known, can make it easier for a hacker to access the computer. It is this type of unauthorised access that I shall investigate, and I shall show what the hacker then did with the computer system.

System Description

The computer system on which the honeypot was installed was a normal Personal Computer. This particular computer system was built by me as a computer system for my work.

The system was installed with a Linux Operating System called *Fedora Core 1*. The honeypot was created using an additional piece of computer software called *VMWare 4.5 Workstation*. This software allowed the honeypot to be contained completely separate from the host computer system, and exist as if it was a computer in its own right. This is referred to as a Virtual Machine, or VM. Inside this virtual machine, a second operating system was installed. This is the operating system that it was hoped a hacker would attack.

The security of the system that contains the honeypot is very important. I did not want this system to be seen by the attacker or they may know the system was not all it seamed. To do this, I utilised a computer program from the people widely recognised as the experts in this field, the Honeynet project. This program is called rc.firewall, and it provides a quick and easy method of ensuring the security of my computer system. rc.firewall also provides a second and equally important feature, it restricts the ability of the hacker to attack other computer systems from the honeypot. This is referred to as downstream liability limitation.

The methods of installing the virtual machine, and the security software were taken from a set of instructions again provided by the Honeynet project. This document is available on the Internet, and is called: **Know Your Enemy: Learning with VMware** located on the Internet here http://www.honeynet.org/papers/vmware/

The system was connected to the Internet via home computer network, a 1Mb ADSL connection. To 'sweeten' the honeypot thereby making the system more interesting to the hacker, an Internet domain name was allocated at the system, and the system added to the online computer survey, *The Netcraft Survey*[5], which is often used by hackers to discover which computer are known to be running services which the hacker knows has errors in it allowing him to attack the system.

From the moment the computer system was connected to the Internet, The system was regularly scanned and probed by many different computers. This continued for eight days until the system was eventually compromised. This report details the forensic investigation to discover what occurred.

Analysis of the system hardware

The computer system being investigated was a personal computer system of the following specification:

- AMD XP2100+ Central Processor Unit (CPU)
- 512Mb Random Access Memory (RAM)
- One hard disk
  - 82.3 GB - 3.5" Hitachi hard disk drive
- Two Ethernet Cards
- Graphics card
- One internal DVD Rom drive

To allow the computer system to be readily identified, as it does not have a serial number, the following photograph was taken and entered into evidence with evidence tag number #00001.

[5] http://www.netcraft.com

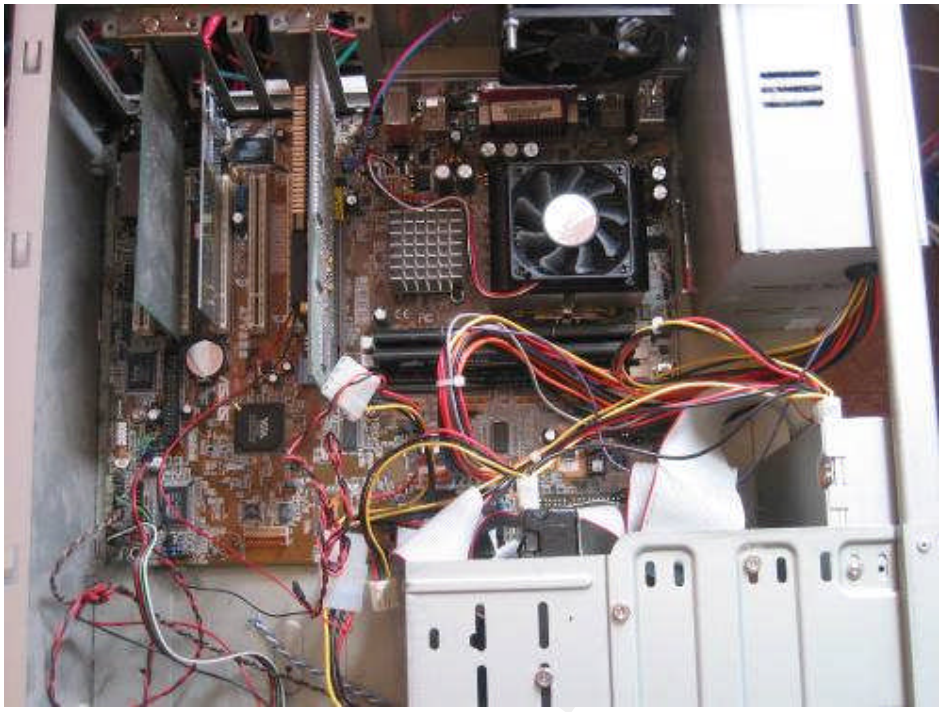


Figure 45 – Internal Identification Photograph of Honeypot System (Tag # 00001)

Figure 46 – Evidence Photo of disk contain VMWare instance (TAG # 00002)

The above photograph was taken of the computer disk drive Serial Number G3CG5H5G which is a Hitachi Deskstar 3.5" hard disk drive of 82.3 GB in capacity. This disk drive held the honeypot.

Tag Numbers	Description of the item	Serial Number
Tag # 00001	**Custom built computer system, no serial number evident Photographic identification taken**	**None Available**
Tag # 00002	**Hitachi Deskstar 3.5" hard disk drive**	**G3CG5H5G**

Forensic Imaging of the suspect system

The content of the computer systems disk drive has to be copied using a technique called *forensic imaging* to allow the investigation to be performed without damaging the original evidence. This is a technique where an exact copy of the information contained on the hard disks of the system is made. To perform this task, an image of the disc drive is copied from one computer system to another computer system on which the investigation was to be conducted.

This was done using a UNIX computer command called 'dd'. This allowed me to read the data from the disk and then, in conjunction with an additional command called 'nc', transfer this image to the computer system on which I was to perform the forensic investigation.

To ensure that the data that was read from the compromised system was exactly the same as the image I was to investigate, a calculation was performed called an *md5 message digest*[6]. This is a technique where a unique number is calculated which individually identifies a computer file. I then compared the md5 value calculated from contents of the computer disk with the md5 value calculated from the file I analysed; the two values must be equal.

In the following two images (Figure 47 and Figure 48), I will show the md5 values calculated. As you can see, they are the same, and therefore, I am able to guarantee that the image I have obtained is a true image of the computer system being investigated.

In a forensic investigation, it is common to remove the disk drive, and image the whole contents to a separate disk drive. This is not possible in this investigation as the computer system is "Virtual" therefore the data has to be copied from the virtual computer to another system.

In the image below you can see the VMWare system running the computer system named *Redhat 7.3*. I have used a CD based suite of programs called *Helix Version 1.4*[7]. This allows me to use programs from the CD to perform the forensic image, rather than from the compromised computer system thereby ensuring I do not contaminate any evidence and also to ensure that I am certain that the commands I will be using are in a known state.

To perform the imaging, the following command was executed on my forensic workstation:

```
# nc –l –p 5555 | dd of=honeypot.sda2.dd
```

[6] http://www.ietf.org/rfc/rfc1321.txt?number=1321
[7] http://www.e-fense.com/helix/

This uses the nc command to listen on connection port 5555 for incoming connections. It then passes the output to dd to write the data to a file called honeypot.sda2.dd. It is this image I will investigate.

The following command was then executed on the Helix system:

```
# dd if=/dev/sda2 | nc 192.168.1.101 5555
```

This uses the command dd again to read the part of the hard disk drive which contains the Operating System I need to image. Here it is referred to as /dev/sda2. The nc command causes the computer to connected to the other computer system at the network address 192.168.1.101 and connection port 5555.

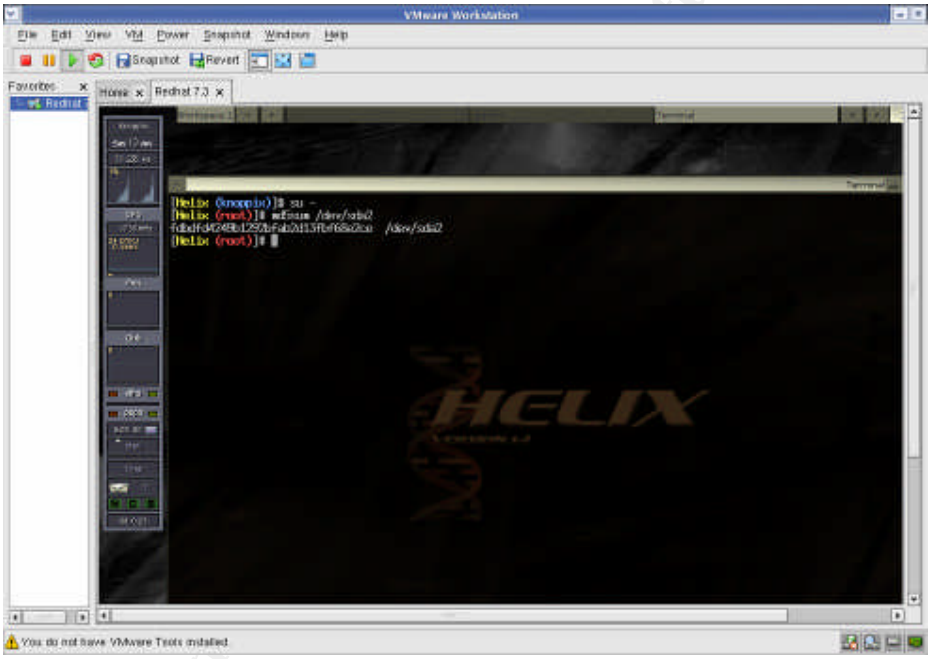


Figure 47 – MD5 value of the honeypot disk drive

Once the forensic imaging has completed, I perform an md5 comparison of the two images to ensure that the copy I have taken is exactly the same as the original. The comparison can be seen my comparing the numbers shown in the image above in Figure 43 and below in Figure 44.

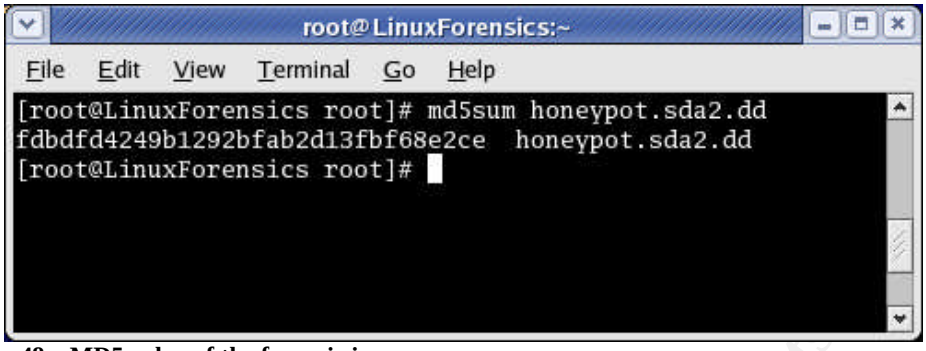


Figure 48 – MD5 value of the forensic image

Media Analysis of System

The obtained image was analysed on my Forensic Workstation. This is a dedicated computer system which has specialist tools installed to perform the analysis.

The Forensic Workstation is comprised of:

- AMD XP2500+
- 512Mb RAM
- Two hard disks
  - 40GB - 3.5" Samsung (SP0411N) hard disk drive
  - 123GB – 3.5" Hitachi (HDS722512VLAT20) hard disk drive
- Two Ethernet Cards
- Graphics card
- One internal DVD Rom drive
- One internal 3.5" floppy disk drive
- Creative Labs Audigy 1 sound card

The two disk drives installed in the system have separate functions. The 40GB Samsung hard disk drive contains the Operating System and tools that together allow me to perform the investigation. The 123GB Hitachi hard disk drive is the system that will hold the evidence. This will allow me to ensure that any evidence from previous investigations is forensically removed before starting a new investigation.

The Operating System installed on the system was Fedora Core 1, which had been updated to the latest release of all its component parts.

The Forensic Analysis software used consisted of:

- The latest release of The Sleuth Kit (Version 1.69)
- The latest release of the Autopsy Browser (Version 2.0)

The Sleuth Kit is the heart of my forensic workstation. It provides the tools so that I can perform the investigation.

The Autopsy Browser provides a simple graphical interface to the tools allowing quicker and often better presented results to be obtained more simply.

All of the software discussed is referred to as Open Source. This means that they have not been developed by a single company for retail purposes, but is often written and reviewed by many developers across the world. The suitability of such software in a court of law is often discussed. The following two references show the suitability of the software in a US court of law:

http://www.cerias.purdue.edu/homes/carrier/forensics/docs/opensrc_legal.pdf

http://www.vjolt.net/vol6/issue3/v6i3-a13-Kenneally.html

The forensic image I copied earlier is imported into the Autopsy Forensic Browser:

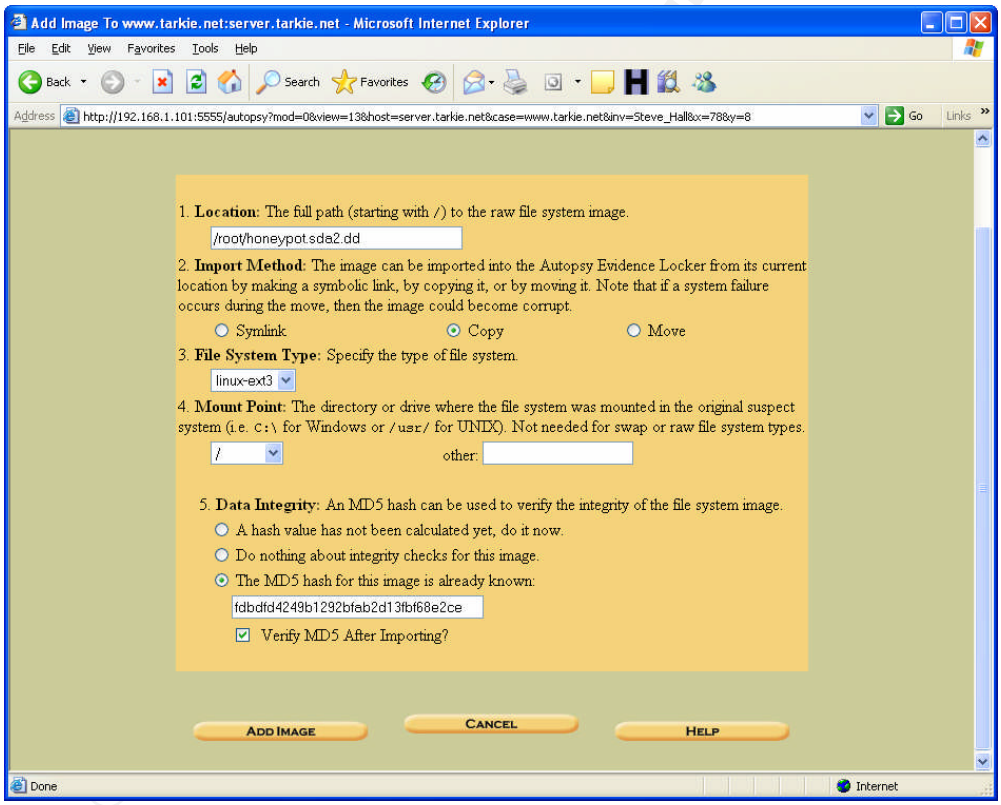


Figure 49 – Importing forensic image into Autopsy

I enter, and select for the MD5 checksum to be verified after the image is copied into the evidence locker. As can be seen below, the values match which confirms that the image imported is identical.

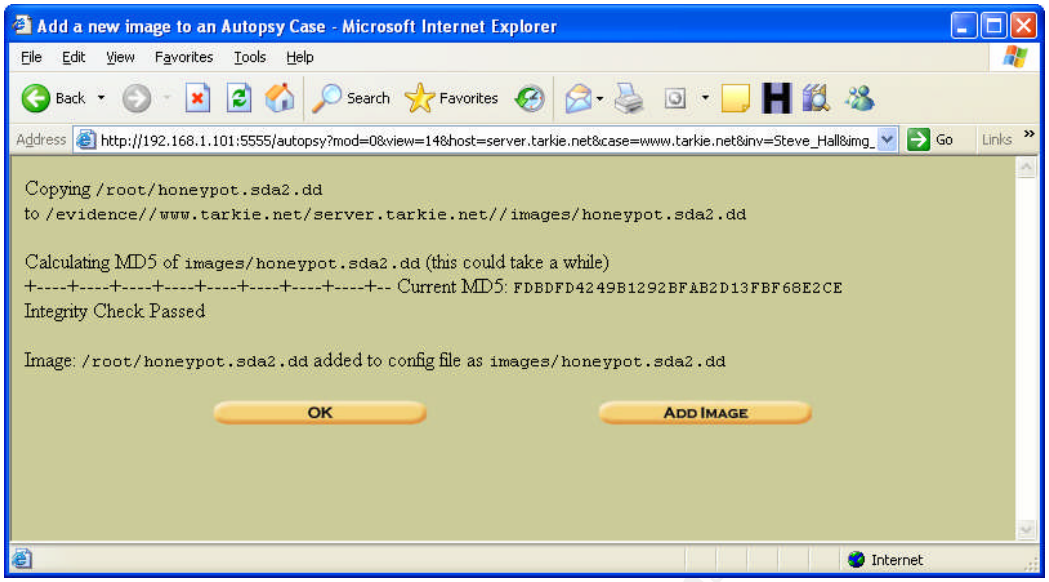


Figure 50 – MD5 checksums are A-ok!

Once I click OK, I am given the option to see the image details for this investigation. I am then able to extract any strings of characters found in the image to allow me to search for key words or phrases, and to extract unallocated disk space for later investigation. I use both options, and each time ensure than an md5 checksum is taken. Once the unallocated disk space has been extracted I get an additional option to extract strings found in this area of disk space. I use this option and once complete the image details screen now shows the following:

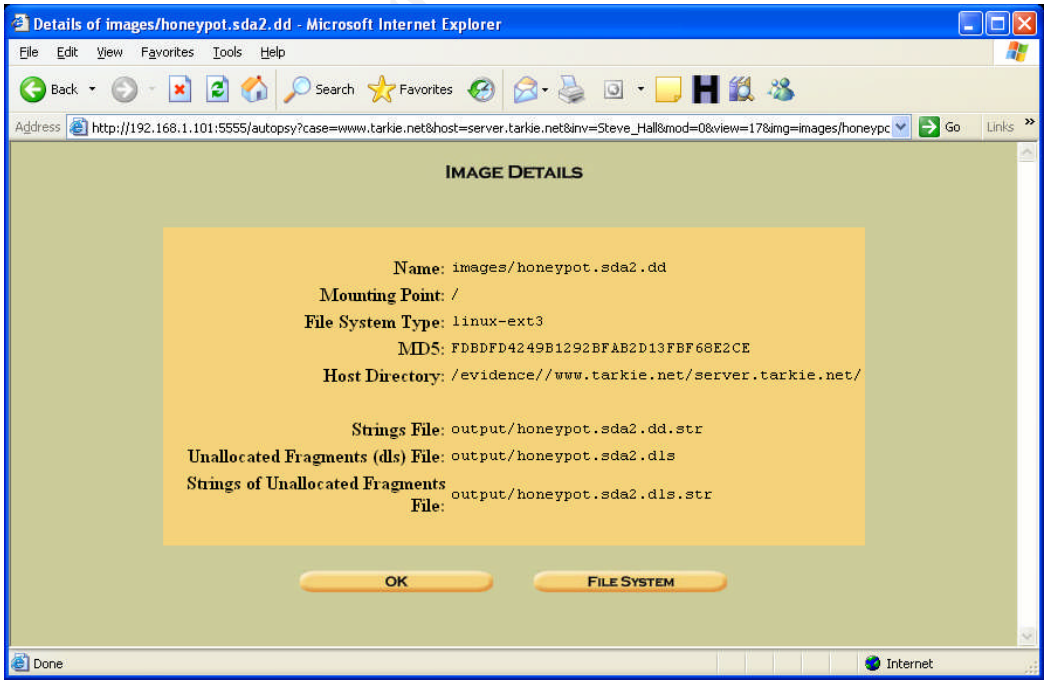


Figure 51 – completed unallocated fragments output and strings

The next stage is to generate a *file access timeline*. This allows me to playback changes to the files so I can see what has happened on the system. The timeline shows when files are created, modified or accessed.

Creating the timeline can be performed a number of ways, utilising a number of tools such as mactime, MAC-Daddy or as here, using the Autopsy browser. The browser initially runs the following commands:

```
fls -r –m
```

using the image file:

```
images/honeypot.sda2.dd
```

The –r causes the file to recurse the directory structure, the –m causes the output to be suitable for mactime analysis

```
ils -m
```

again using the image file

```
images/honeypot.sda2.dd
```

The –m causes the output to be suitable for mactime analysis.

This creates a 'body' file, which is then used to create the timeline using a tool called mactime. The mactime takes the body file, the password and group file locations and outputs a date sequenced file which shows file system activity.

During this investigation, I have created the timeline using the autopsy browser. Once successfully created, the following is displayed:

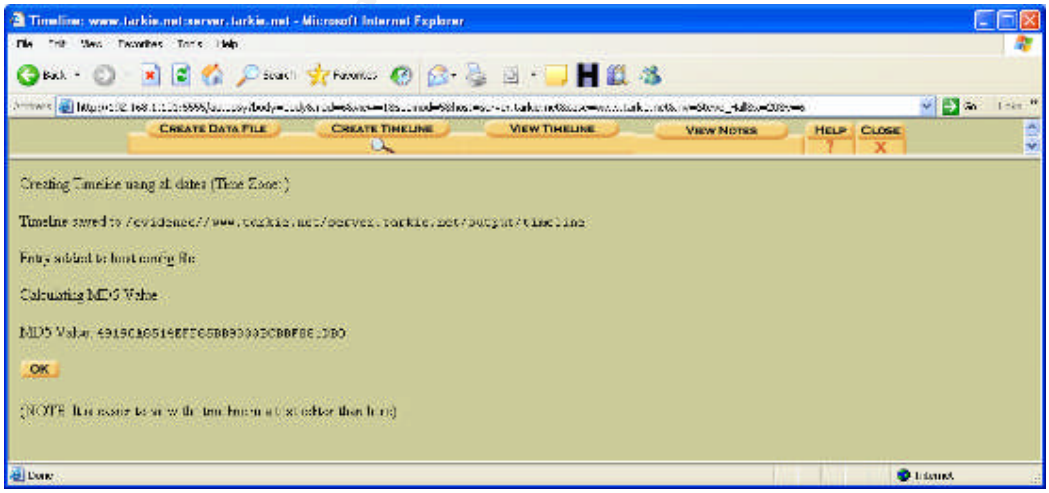


Figure 52 – successful creation of the timeline file

Important information can be gained from using the summary screen provided as part of the browser timeline interface. As an example of what you can use it for below you can see two large numbers representing the number of timeline entries for the days 30th May 2004 which was the day of installation of the honeypot and 5th June 2004 the day the honeypot was compromised.

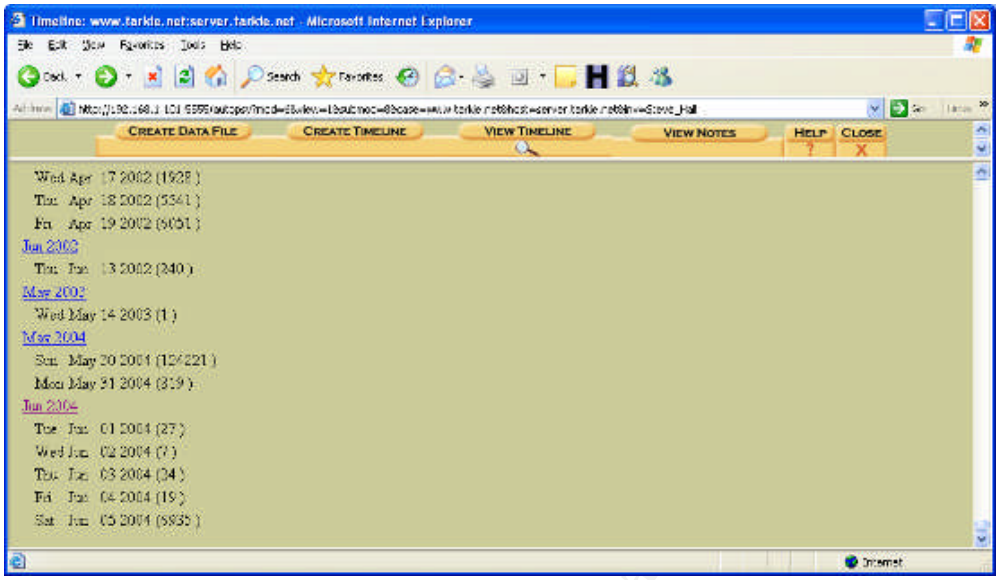


Figure 53 – sample timeline summary within Autopsy

A closer look at the Sunday, May 30th entries shows the whole of the installation process. Some important entries of this are shown below; firstly the creation of the basic directory structure and the install log begins:

```
Sun May 30 2004 20:52:59    16384 m.c d/drwx------ root     root     11
/lost+found
                                0 mac ---------- root     root     1
<honeypot.sda2.dd-alive-1>
Sun May 30 2004 20:53:07     4096 mac d/drwxr-xr-x root     root     97729    /dev/pts
                             4096 mac d/drwxr-xr-x root     root     130305   /proc
                             4096 mac d/drwxr-xr-x root     root     32577    /boot
Sun May 30 2004 20:53:17        0 mac -/-rw-r--r-- root     root     293187
/root/install.log.syslog
                            22057 .a. -/-rw-r--r-- root     root     293186
/root/install.log
```

Note: the complete timeline is available here:

http://homepages.nildram.co.uk/~tarkie/GIAC/GCFA/timeline.rar

To detect attacks on the honeypot, an additional software tool was used called an Intrusion Detection System (IDS), this watches the network for known attack signatures. The IDS system deployed in this investigation was the SNORT IDS[8] using the latest available ruleset at the time of deployment. This was deployed on the system which hosted the VMWare honeypot so it was undetectable to the attacker. The Intrusion Detection system issued an alert and, as shown below, this gives me an estimated time that the intrusion took place.

[8] http://www.snort.org

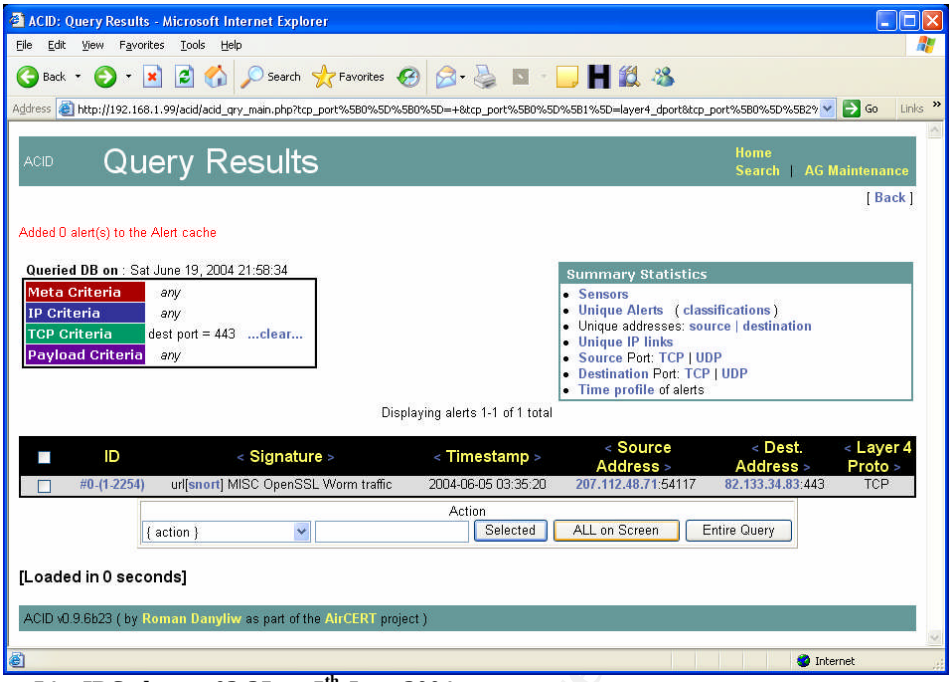


Figure 54 – IDS alert at 03:35am 5th June 2004

To reinforce this point in time as the moment of intrusion the following IDS entry was also found at this time:

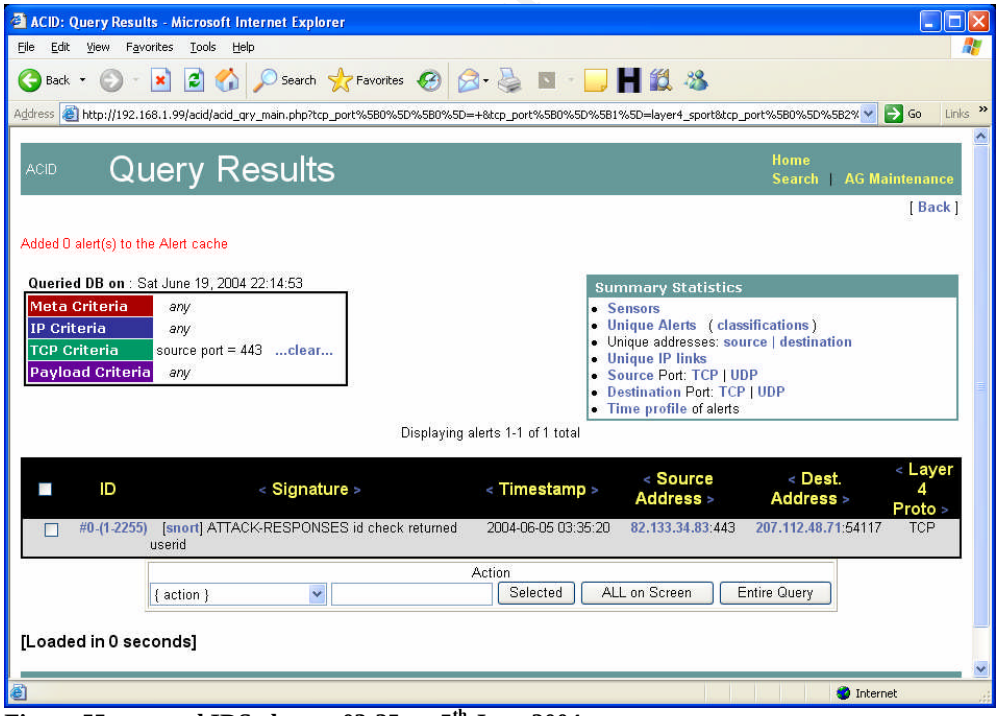


Figure 55 – second IDS alert at 03:35am 5th June 2004

This shows that the output from the Linux command *'id'* was sent back to the IP address 207.112.48.71. This command is often used to discover what user the hacker has obtained via the exploit on the target system.

The timeline at this time shows some very interesting entries. I shall start my investigation from this point in time.

As shown below in Figure 56, the web server log file; /var/log/httpd/ssl_engine_log has been updated. Shortly after this a number of steps happen:

1. The /var/tmp directory is accessed
2. The file newtrace is created in /var/tmp
3. The whoami command is executed
4. The /etc/issue file is accessed
5. A file, now deleted, is accessed
6. The telnet daemon startup script is modified
7. The killall command is executed

From this I see the first signs of our intrusion, and I have the first items to investigate; what is *newtrace*, what happened to the telnet daemon, and why was *killall* run.

In addition to this, I can also see the first sign of the honeypot being prepared for long term occupation. At 03:39:50 a file called 1login.tgz is created in a hidden directory (/dev/.tty). It would appear that this could be a replacement for the login program being prepared for use as unauthorised back door entry points to the computer. This would allow unrestricted access at a later date should the initial vulnerability used to access the computer be fixed. And from the output of the two IDS alerts I have a network address to investigate: 207.112.48.71

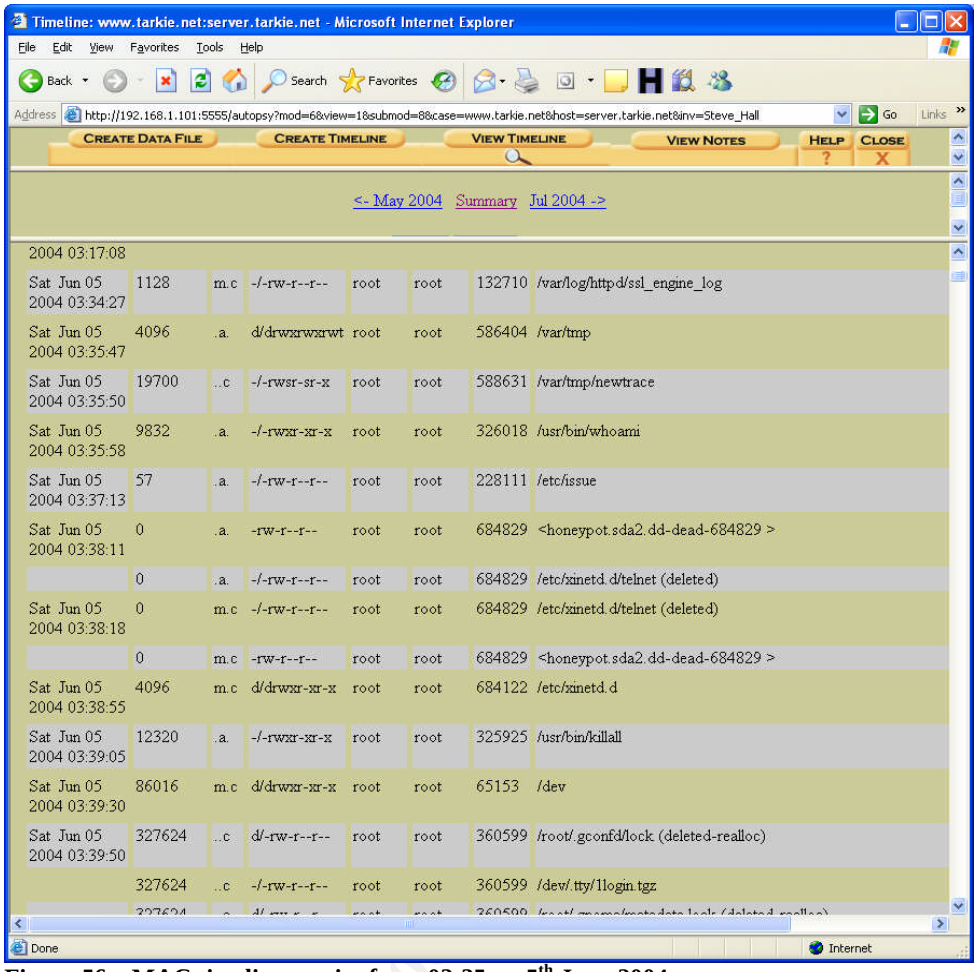


Figure 56 – MAC timeline entries from 03:35am 5th June 2004

Further analysis of the hidden directory (/dev/.tty) provides me with the following list of files installed in that directory:

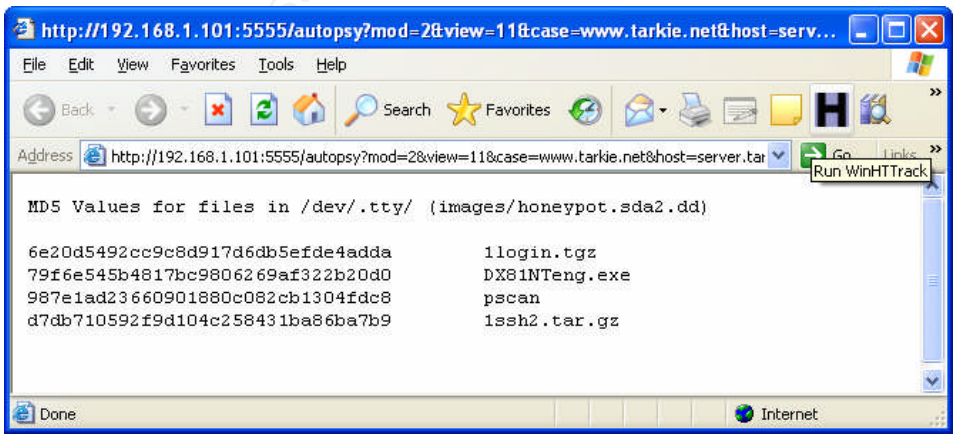


Figure 57 – Contents of files in /dev/.tty

It would appear that I have located two replacement commands; login and ssh2 and interestingly, the attacker appear to have downloaded a copy of DirectX for Microsoft Windows 2000 or Windows XP in the directory, and also

an unknown file called pscan. It is possible therefore, that these replacement commands are Trojan versions and this needs to be confirmed.

The /var/tmp directory was accessed, and an unknown file was found there called newtrace. Further analysis of the directory has found the following, as shown in the screenshot shown in Figure 59 below.

A further unknown file called *BDB011102* had been created but has since been deleted. The contents of the file are still available, and the ASCII strings contained within the file are shown.

The strings indicate that this is likely to be a file normally called dialchk.c version 1.6. Searching for this version of the software via Google, I find that dialchk.c appears to be part of the Shadow project program. The following is the description of the shadow project:

> "The Shadow password file utilities package includes the programs necessary to convert traditional V7 UNIX password files to the SVR4 shadow password format, and additional tools to maintain password and group files (that work with both shadow and non-shadow passwords).[9]"

[9] http://freshmeat.net/projects/shadow/

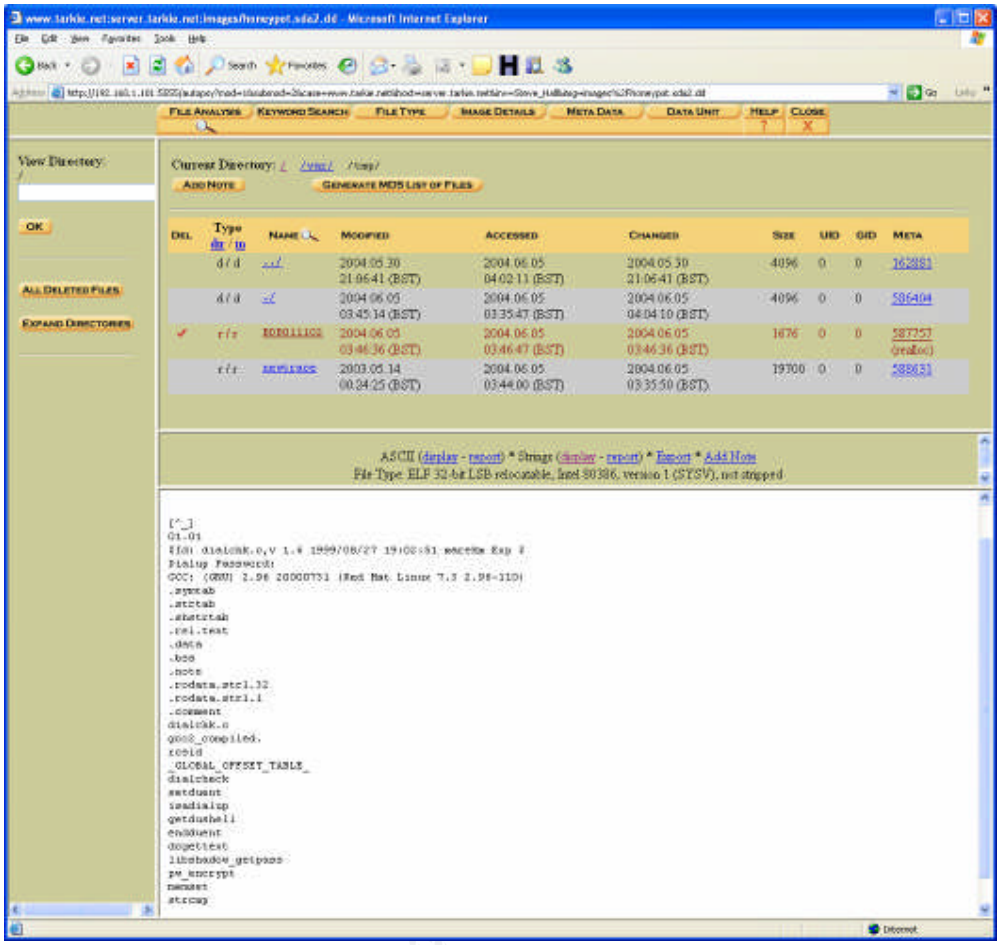


Figure 58 - Examination of a deleted file

Moving on to the telnet daemon, I found that the start up script located in /etc/xinetd.d/telnet had been interestingly modified. The Apache web server that the attack was launched via was running both HTTP and SSL web servers. The attack was launched via the SSL connection which is achieved over TCP port 443. The modification of the telnet daemon was to enable a remote telnet connection on port 443:

```
Contents Of File: /etc/xinetd.d/telnet

service telnet
{
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
        log_on_failure  += USERID
          protocol  = tcp
          port              = 443
}
```

Interestingly, the first indication that the honeypot had been hacked was not from my IDS, but from a friend. Before checking my IDS logs on the morning of the hack, an instant message was waiting for me:

```
Session Start (XXXXXXXX:Fish out o water): Sat Jun 05 04:04:08 2004
Fish out o water: So what happened to SSL support on your Apache
webserver !?!?!?
*** "Fish out o water" signed off at Sat Jun 05 05:18:04 2004.
```

I now have an indication as to:

- how the attacked got root access to the system
- how access was kept to the system

I need to dig deeper into how the server was compromised and also to find what the attacker then used the system for.

How the server was compromised

As shown earlier, I know that the IDS system alerted "MISC OpenSSL Worm traffic". This is detailed in more depth within the CERT Advisory CA-2002-27[10]. The IDS signature which trigged this alert is shown below:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"MISC OpenSSL Worm traffic";
flow:to_server,established; content:"TERM=xterm"; nocase;
reference:url,www.cert.org/advisories/CA-2002-27.html; classtype:web-application-
attack; sid:1887; rev:3;)
```

The key to this signature is that the content string "TERM=xterm" (highlighted) is sent to an SSL web server, which is certainly not to be considered as normal traffic as an xterm is a graphical terminal for access to a system.

Looking at the apache ssl_engine_log, I can also see the failure which allowed the attacker access to the system:

```
[05/Jun/2004 03:34:27 09342] [error] SSL handshake failed (server
server.xxxxxxx.net:443, client 207.112.48.71) (OpenSSL library error follows)
[05/Jun/2004 03:34:27 09342] [error] OpenSSL:
error:1406908F:lib(20):func(105):reason(143)
```

Again, taking a key phrase from the log file, and searching for the "*OpenSSL:error:1406908F:lib"* string on Google I found information which potentially identifies the exploit used as the OpenSSL-too-open exploit[11].

Checking back through the Apache web server logs for the IP address 207.112.48.71 I find the following single entry in the access logs for the HTTP web server.

```
207.112.48.71 - - [05/Jun/2004:03:33:43 +0100] "GET /sumthin HTTP/1.0" 404 275 "-" "-"
```

[10] http://www.cert.org/advisories/CA-2002-27.html

[11] http://packetstormsecurity.org/0209-exploits/openssl-too-open.tar.gz

Again, Google comes to the rescue with a possible answer. Searching for "*GET /sumthin HTTP/1.0*" results in a large number of hits with administrators asking what this was. One post however may give the answer as the ATD OpenSSL Mass Exploiter[12]. An analysis of this tool can be found on the computer security site LURHQ[13].

What the attacker used the system for

Further analysis of the timeline shows the following entries:

```
# fgrep "/dev/.tty/pscan" timeline
                          17325 m.. -/-rwxr-xr-x root     root     359724
/dev/.tty/pscan
Sat Jun 05 2004 05:17:10    17325 ..c -/-rwxr-xr-x root     root     359724
/dev/.tty/pscan
                          17325 .a. -/-rwxr-xr-x root     root     359724
/dev/.tty/pscan
```

By looking at the timeline entries which are recorded at nearly the same time as these I can see that the /dev/.tty/pscan was accessed at exactly the same time as the following entry:

Sat Jun 05 2004 05:17:38	17325	m..	-/-rwxr-xr-x	root	root	670080	/dev/.tty/newssh/pscan
	17325	.a.	-/-rwxr-xr-x	root	root	359724	/var/spool/mqueue/xfi55324f14732 (deleted-realloc)
	17325	.a.	-/-rwxr-xr-x	root	root	359724	/dev/.tty/pscan

It would appear from this that the temporary mail spool file /var/spool/mqueue/xfi55324f14732 may have been created when pscan was run. The mail spool file was deleted, but I may be able to recover the contents of the file from the Autopsy Forensic Browser:

[12] http://www.webmasterworld.com/forum11/2100.htm

[13] http://www.lurhq.com/atd.html

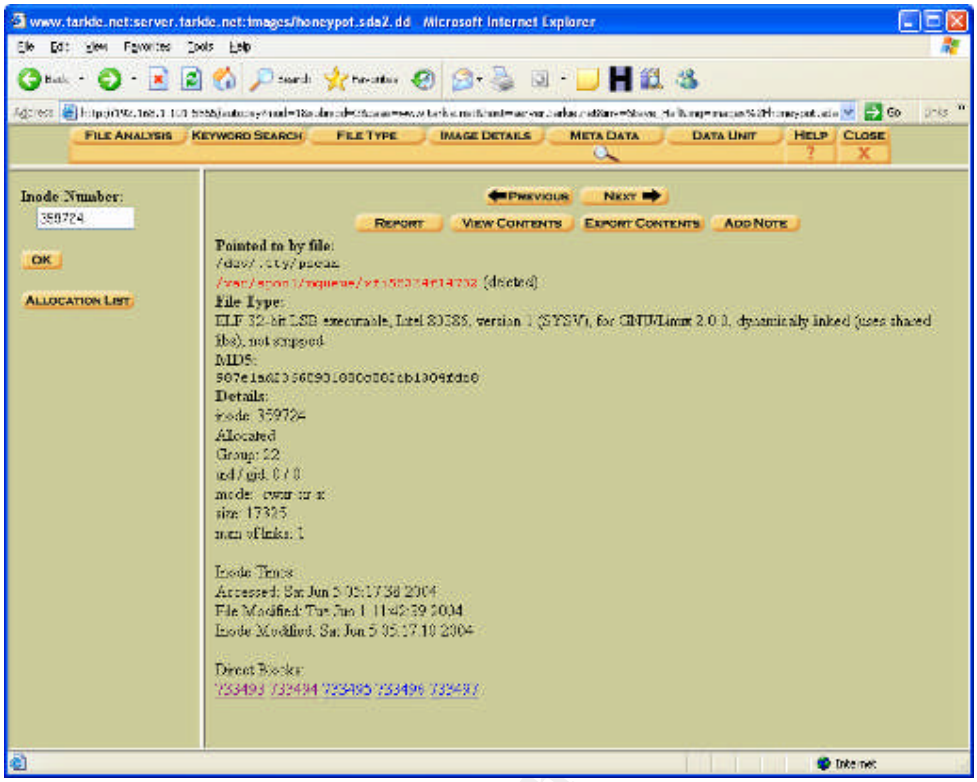


Figure 59 – Identification of a deleted mail spool file

Unfortunately it would appear that the mail spool file has been overwritten by the /dev/.tty/pscan binary, as when I try and recover the file, the pscan binary is recovered.

It should be noted at this point that recovering a file based on the inode number from an ext3 file system is normally impossible. The ext3 *Frequently Asked Questions* guide states:

"In order to ensure that ext3 can safely resume an unlink after a crash, it actually zeros out the block pointers in the inode, whereas ext2 just marks these blocks as unused in the block bitmaps and marks the inode as "deleted" and leaves the block pointers alone.

Your only hope is to "grep" for parts of your files that have been deleted and hope for the best."[14]

In the directory /dev/.tty/newssh more is found than is expected. Not only does the directory hold a suspected replacement for ssh2d but a file called targets. This file contains a number of lines with what appears to be version information relating to specific ssh versions.

[14] http://batleth.sapienti-sat.org/projects/FAQs/ext3-faq.html.

```
Small - SSH-1.5-
1.2.25,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-
1.2.25,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-
1.2.26,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-
1.2.26,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-
1.2.27,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-
1.2.27,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-
1.2.30,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-
1.2.30,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-
1.2.31,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-
1.2.31,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-OpenSSH-
1.2,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-OpenSSH-
1.2,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.5-OpenSSH-
1.2.2,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,0
Big - SSH-1.5-OpenSSH-
1.2.2,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a,0x0805,1
Small - SSH-1.99-
OpenSSH_2.2.0p1,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a
,0x0805,0
Big - SSH-1.99-
OpenSSH_2.2.0p1,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a
,0x0805,1
Big - SSH-1.99-
OpenSSH_2.2.0p1,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a
,0x0805,1
Small - SSH-1.99-
OpenSSH_2.5.2p2,0x08070000,0x08184000,0x00000004,0x00010004,0x00000000,0x08400000,0x7a
,0x0805,1
```

Further analysis of the files in this directory results in a piece of 'C' code being recovered. It is a fast scanner for Secure Shell Daemon banner strings. This code can scan upto 200 systems at the same time. This is shown below:

```
// made by asssaf and y4nir //
// #^247^232^236^240^233 team on DALnet //

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/types.h>

#define CONNECT_TIMEOUT               20
#define MAX_CHILDREN                  200

pid_t wait(int *status);

int sock;

void connect_read_timeout() {
         close(sock);
}

void checkout(struct in_addr ip) {
    FILE *etog; char nuf[1024], line[1024];
```

```
    int found=0;
    struct sockaddr_in sa;

    memset(&sa, 0, sizeof(sa));
    memcpy(&sa.sin_addr, &ip, 4);
    sa.sin_port = htons(22);
    if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
         exit(0);

    signal(SIGALRM, connect_read_timeout);
    alarm(CONNECT_TIMEOUT);
    if (connect(sock, (struct sockaddr *)&sa, sizeof(sa)) == -1) {
         alarm(0); exit(0);
    } else {
         recv(sock, nuf, 1024, 0);
        if(nuf[strlen(nuf) - 1] == '\n')
             nuf[strlen(nuf) - 1] = '\0';


                  if(strstr(nuf, "SSH-1.5-1.2.25")) {
         snprintf(line, sizeof(line), "%s: hackable 1 or 2\n", inet_ntoa(ip));
         found++;
         }
                           if(strstr(nuf, "SSH-1.5-1.2.26")) {
         snprintf(line, sizeof(line), "%s: hackable 3 or 4\n", inet_ntoa(ip));
         found++;
         }
                                    if(strstr(nuf, "SSH-1.5-1.2.27")) {
         snprintf(line, sizeof(line), "%s: hackable 5 or 6\n", inet_ntoa(ip));
         found++;
         }
                                              if(strstr(nuf, "SSH-1.5-1.2.30")) {
         snprintf(line, sizeof(line), "%s: hackable 7 or 8\n", inet_ntoa(ip));
         found++;
         }
                                                        if(strstr(nuf, "SSH-1.5-
1.2.31")) {
         snprintf(line, sizeof(line), "%s: hackable 9 or 10\n", inet_ntoa(ip));
         found++;
         }
         if(strstr(nuf, "SSH-1.5-OpenSSH-1.2")) {
         snprintf(line, sizeof(line), "%s: hackable 11 or 12\n", inet_ntoa(ip));
         found++;
         }
                  if(strstr(nuf, "SSH-1.5-OpenSSH-1.2.2")) {
         snprintf(line, sizeof(line), "%s: hackable 13 or 14\n", inet_ntoa(ip));
         found++;
         }
                           if(strstr(nuf, "SSH-1.99-OpenSSH_2.2.0p1")) {
         snprintf(line, sizeof(line), "%s: hackable 15 or 16 or 17\n",
inet_ntoa(ip));
         found++;
         }
                           if(strstr(nuf, "SSH-1.99-OpenSSH_2.2.0p1")) {
         snprintf(line, sizeof(line), "%s: hackable 15 or 16 or 17\n",
inet_ntoa(ip));
         found++;
         }
         if(strstr(nuf, "SSH-1.99-OpenSSH_2.5.2p2")) {
         snprintf(line, sizeof(line), "%s: hackable 18\n", inet_ntoa(ip));
         found++;
         }
         if(found) {
         etog = fopen("bla.log", "aw+");
         fprintf(etog, line);
         fclose(etog);
         }
         }
         alarm(0);
         exit(0);
}

int main(int argc, char *argv[]) {
char buf[100];
FILE *buu; int childs = 0;
struct in_addr ip;
```

```
if(argc < 2) {
    fprintf(stderr, "Scanner by asssaf and Y4nir (ema@zona.com)\n\n");
    fprintf(stderr, "Usage: %s <IP List>\n", argv[0]);
    exit(0);
}

if((buu = fopen(argv[1], "r")) == NULL) { perror(argv[1]); exit(0); }
while(fgets(buf, sizeof(buf), buu) != NULL) {

    if(buf[strlen(buf) - 1] == '\n')
         buf[strlen(buf) - 1] = '\0';

    if(inet_aton(buf, &ip) != 0 && ip.s_addr != 0) {
         if(childs >= MAX_CHILDREN) wait(NULL);
         switch (fork()) {
             case 0:
                       checkout(ip);
                       exit(0);
             case -1:
                       perror("fork");
                       exit(-1);
             default:
                       childs++;
                       break;
} } else { while(childs--) wait(NULL); } } return; }

// made by asssaf and y4nir //
// #^247^232^236^240^233 team on DALnet //
```

The MAC Timeline shows that this code was created, along with the other files in the directory on Saturday, June 5th 2004 at 05:17:44 that it was compiled on the honeypot on Saturday, June 5th 2004 at 05:24:39 and executed on Saturday, June 5th 2004 at 07:09:02.

Analysis of the ASCII strings contained within the file /dev/.tty/newssh/pscan shows the following text:

```
Usage: %s <b-block> <port> [c-block]
Invalid b-range.
Unable to allocate socket.
Unable to set O_NONBLOCK
%s.%d.%d
Invalid IP.
Scan completed in %u seconds.
%15s - %2u second(s)
Error: %s
echo %s >> open.log
```

It would appear that pscan is a port scanner which scans for a specified port number and outputs the results to a file called open.log. The directory /dev/.tty/newssh contains a file called open.log. The following is an extract from this log:

```
212.202.0.17
212.202.0.50
212.202.0.65
212.202.0.87
212.202.0.154
212.202.0.204
212.202.0.224
212.202.3.253
212.202.4.30
212.202.4.109
212.202.5.41
```

It would appear that the hacker scanned the Class-B network 212.202.0.0 for known vulnerable Secure Shell servers and stored the output in open.log. The potential for this scan is serious, as the attacker has a very large list of potentially vulnerable servers to compromise.

At this point in the investigation, an abuse e-mail was sent to the owners of the 212.202.0.0 network address space reporting the events. This was sent to abuse@qsc.de

A total of seven class B networks were scanned in total.

To check if the hacker has any further lists of networks to be scanned, the text search for IP addresses is performed within Autopsy. This results in the evidence that open.log exists in two places the second being /var/www/html/open.log.

From the timeline, it was also possible to see that open.log was also created in /var/www/html/ thus allowing the results of the scanning to be downloaded from the same webserver as allowed the intrusion. In addition to this, the timeline shows that the file was potentially downloaded at 07:22:23.

Sat Jun 05 2004 07:22:23	51568	.a.	-/-rw-r--r--	root	root	571152	/var/www/html/open.log

The MAC timeline will show the last time the file was access, to find all the times the apache web server logs are required:

```
130.89.163.48 - - [05/Jun/2004:05:30:53 +0100] "GET /open.log HTTP/1.0" 200 14596 "-"
"Wget/1.5.3"
130.89.163.48 - - [05/Jun/2004:05:44:56 +0100] "GET /open.log HTTP/1.0" 200 21580 "-"
"Wget/1.5.3"
64.68.82.184 - - [05/Jun/2004:06:11:57 +0100] "GET /robots.txt HTTP/1.0" 404 278 "-"
"Googlebot/2.1 (+http://www.googlebot.com/bot.html)"
64.68.82.184 - - [05/Jun/2004:06:11:59 +0100] "GET / HTTP/1.0" 304 - "-"
"Googlebot/2.1 (+http://www.googlebot.com/bot.html)"
130.89.163.48 - - [05/Jun/2004:06:12:40 +0100] "GET /open.log HTTP/1.0" 200 18547 "-"
"Wget/1.5.3"
130.89.163.48 - - [05/Jun/2004:06:40:20 +0100] "GET /open.log HTTP/1.0" 200 36662 "-"
"Wget/1.5.3"
130.89.163.48 - - [05/Jun/2004:07:03:40 +0100] "GET /open.log HTTP/1.0" 200 16113 "-"
"Wget/1.5.3"
130.89.163.48 - - [05/Jun/2004:07:03:54 +0100] "GET /open.log HTTP/1.0" 200 16113 "-"
"Wget/1.5.3"
130.89.163.48 - - [05/Jun/2004:07:22:24 +0100] "GET /open.log HTTP/1.0" 200 51568 "-"
"Wget/1.5.3"
```

From this I can see that the IP address 130.89.163.48 downloaded the log file, and that wget was used. The IP address resolves to the Universiteit Twente:

```
Name:    fish.student.utwente.nl
Address:  130.89.163.48
```

The Trojan within

The 1login.tgz file analysis within the MAC Timeline shows some interesting activity in that the file was:

- Created
- Extracted
- Compiled
- Installed

The creation date of 1login.tgz was shortly after the time the IDS system alerted to the intrusion (03:35):

Sat Jun 05 2004 03:39:50	327624	..c	d/-rw-r--r--	root	root	360599	/root/.gconfd/lock (deleted-realloc)
	327624	..c	-/-rw-r--r--	root	root	360599	/dev/.tty/1login.tgz
	327624	..c	d/-rw-r--r--	root	root	360599	/root/.gnome/metadata.lock (deleted-realloc)

It is clear therefore, that from the initial attack that the hacker intended to keep access to the honeypot for as long as possible as it was one of the first tasks to upload the trojaned command.

The compress TAR archive was expanded almost immediately:

Sat Jun 05 2004 03:39:56	3695	..c	-/-rw-r--r--	500	root	556094	/dev/.tty/login/libmisc/setugid.c
	6526	..c	-/-rw-r--r--	500	root	556087	/dev/.tty/login/libmisc/obscure.c
	11100	..c	-/-rw-r--r--	500	root	556105	/dev/.tty/login/libmisc/utmp.c
	1337	..c	-/-rw-r--r--	500	root	556090	/dev/.tty/login/libmisc/pwdcheck.c

As shown below, the config.cache file is created and modified; this would appear to be the time that the Trojan software is started to be compiled. This file is created as the output to the configure command which precedes most source distribution installations.

Sat Jun 05 2004 03:44:45	7529	m.c	-/-rw-r--r--	500	root	473697	/dev/.tty/login/config.cache

As shown below, the gcc C compile is called, and the config information is read to start the build of the source file

Sat Jun 05 2004 03:46:20	3	.a.	l/lrwxrwxrwx	root	root	327596	/usr/bin/cc -> gcc
	24171	.a.	-/-rwxr-xr-x	500	root	473686	/dev/.tty/login/config.sub
	31247	.a.	-/-rwxr-xr-x	500	root	473684	/dev/.tty/login/config.guess

Once compiled, the new Trojan file is installed:

Sat Jun 05 2004 03:47:01	120860	.a.	-/-rwxr-xr-x	root	root	326150	/usr/bin/make
	4096	m.c	d/drwxr-xr-x	root	root	244344	/bin
	4096	m.c	d/drwxr-xr-x	500	root	116282	/dev/.tty/login/src
	0	m.c	-rwxr-xr-x	root	root	244459	<honeypot.sda2.dd-dead-244459 >
	11718	.a.	-/-rw-r--r--	root	root	473701	/dev/.tty/login/Makefile
	70762	..c	-/-r-s--x--x	root	root	116288	/dev/.tty/login/src/login (deleted-realloc)
	70762	..c	-/-r-s--x--x	root	root	116288	/bin/login
	43496	.a.	-/-rwxr-xr-x	root	root	244400	/bin/mv

Analysis of the 1login.tgz contents gives results quickly. The file /dev/.tty/login/README was very informative.

```
Contents Of File: //dev/.tty/login/README

Well, this is pretty straight forward,
edit rk.h, change MY_PASSWORD,
after you install this, you can login with: rewt / MY_PASSWORD (whatever you defined
it to)

also, you need to change MY_LOGFILE
every time a user logs in (not rewt) it will write his username & password into
MY_LOGFILE..
elite eh ?!?

 ! . spwn . !
```

So, I check rk.h (rk may be shorthand for rootkit)

```
Contents Of File: //dev/.tty/login/rk.h

#define MY_LOGFILE "/dev/ttypz"
#define MY_PASSWORD "ohadneu"
```

This highlights a device file created as a logfile which would contain harvested usernames and passwords. However, this file is never created as it does not exist in the timeline, presumably as I did not log into the honeypot once it had been hacked.

Investigation of system logs

The directory /var/log contains many system logs. One of these */var/log/secure* holds logs of security related events. By examining this file I can see some important events:

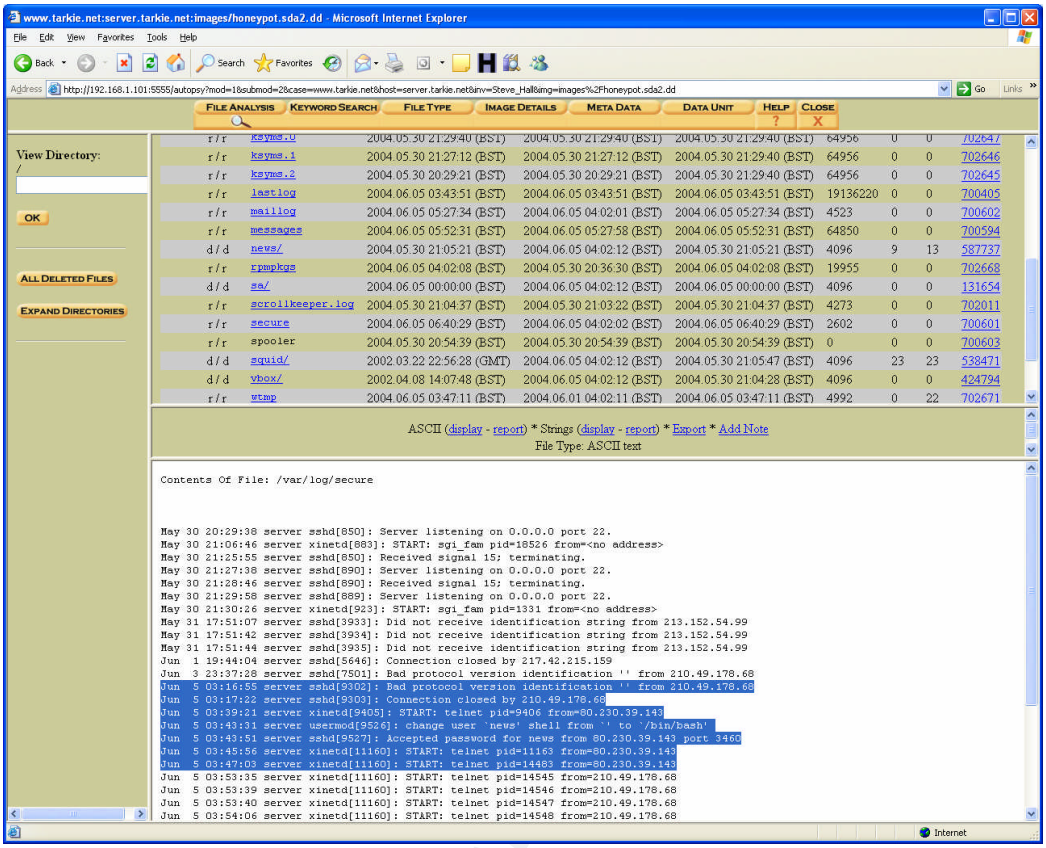


Figure 60 – Analysis of a system log file

At 03:39:21 the telnet service is started from the xinetd service (PID 9405). This would correspond with the *killall* command being seen in the timeline earlier in the investigation. The IP address 80.230.39.143 is used to connect to the system. If I again perform an nslookup on this IP address I get some potentially important information:

```
[root@LinuxForensics root]# nslookup 80.230.39.143
Note:  nslookup is deprecated and may be removed from future releases.
Consider using the `dig' or `host' programs instead.  Run nslookup with
the `-sil[ent]' option to prevent this message from appearing.
Server:         XXX.XXX.34.81
Address:        XXX.XXX.34.81#53

Non-authoritative answer:
143.39.230.80.in-addr.arpa      name = tony03-39-143.inter.net.il.

Authoritative answers can be found from:
39.230.80.in-addr.arpa  nameserver = dns.inter.net.il.
39.230.80.in-addr.arpa  nameserver = dns2.inter.net.il.
dns.inter.net.il        internet address = 192.116.202.99
dns2.inter.net.il       internet address = 192.116.192.9
```

Figure 61 – A name recovered?

It is unusual to have a common name such as Tony recovered from an nslookup unless the IP address issued is a static address and not one that is dynamically issued. This address resolves to a system in Israel.

Unfortunately, by searching for inter.net.il, and tony03 it quickly becomes apparent that this is a red herring. The inter.net.il domain is registered to:

```
domain:         inter.net.il
descr:          Internet Gold
descr:          1 Alexander Yanai St.
descr:          Petach-Tikva
descr:          49277
descr:          Israel
phone:          +972 3 9399891
fax-no:         +972 3 9399700
e-mail:         abuse@zahav.net.il
```

Six minutes after this system connected to the telnet service, a second system connected. This is IP address 210.49.178.69. This address resolves to an ISP in Australia, however this was later discovered to be from the friend sending the instant message noted earlier.

```
[root@LinuxForensics root]# nslookup 210.49.178.68
Note:  nslookup is deprecated and may be removed from future releases.
Consider using the `dig' or `host' programs instead.  Run nslookup with
the `-sil[ent]' option to prevent this message from appearing.
Server:         XXX.XXX.34.81
Address:        XXX.XXX.34.81#53

Non-authoritative answer:
68.178.49.210.in-addr.arpa      name = c210-49-178-68.brasd1.vic.optusnet.com.au.

Authoritative answers can be found from:
178.49.210.in-addr.arpa nameserver = ns1.optusnet.com.au.
178.49.210.in-addr.arpa nameserver = ns2.optusnet.com.au.
ns1.optusnet.com.au     internet address = 203.2.75.2
ns2.optusnet.com.au     internet address = 203.2.75.12
```

Figure 62 – Unknown connection

IP address to investigate

As I have shown earlier, just because an IP address is captured this does not implicate the owner. However, the IP address 207.112.48.71 was captured in a number of places including the IDS system and the web server log files. The IP address can be converted to the name of the system using it by the command *nslookup* as shown below:

```
[root@LinuxForensics root]# nslookup 207.112.48.71
Note:  nslookup is deprecated and may be removed from future releases.
Consider using the `dig' or `host' programs instead.  Run nslookup with
the `-sil[ent]' option to prevent this message from appearing.
Server:         XXX.XXX.34.81
Address:        XXX.XXX.34.81#53

Non-authoritative answer:
71.48.112.207.in-addr.arpa      name = dsl-207-112-48-71.tor.primus.ca.

Authoritative answers can be found from:
48.112.207.in-addr.arpa nameserver = ns1.primus.ca.
48.112.207.in-addr.arpa nameserver = ns2.primus.ca.
```

Figure 63 – identification of the hackers system?

The system is connected to a DSL connection in Canada. Primus appears to be a telecoms company within Canada:

```
Domain Name: primus.ca
Registered: 2000/10/25

Last Modified: 2004/01/29
Expires: 2006/02/22

Registrant: PRIMUS Telecommunications Canada Inc.
Carl Scase dns-house@primustel.ca
416-236-3636 FAX 416-207-7169
```

It is possible however, that this IP address is dynamically allocated to primus.ca's users each time they connect. Therefore, any further investigation of this IP address is potentially erroneous.

The Internet site, www.mynetwatchman.com[15] correlates intrusion attempts on a global scale. By searching for our target IP address, I get a hit. The event logged looks very similar to this event, in that it was launched over an SSL connection, and was logged the day before.

Most Recent Event									
Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target IP	# of IPs Targeted	Protocol/ Port	Port/ Issue Description	Source Port	Event Count
4 Jun 2004 09:32:46	sburina	Perl	Cisco Rtr	81.15.x.x	1	6/443	HTTPS - HTTP over TLS/SSL HTTPS - HTTP over TLS/SSL	56197	3

The hidden directory

During the compromise, the attacker created a hidden directory. This utilised a function of the Linux operating system that hides files and directories for normal view by prefixing them with a dot. The directory in this instance is called */dev/.tty*. The attacker selected the /dev/ directory to hide this as normally it has many thousands of files located in it. The /dev/.tty directory contains the following files:

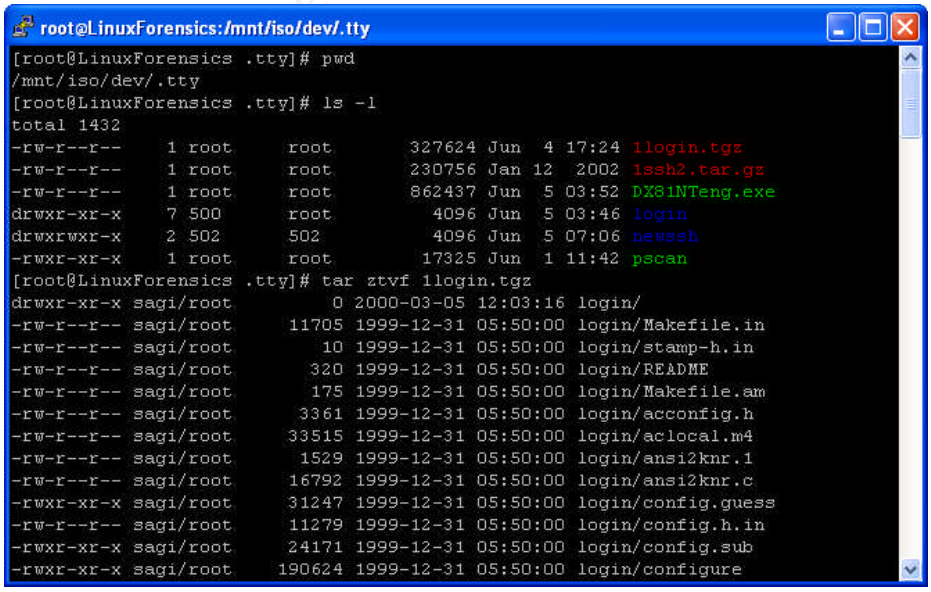


Figure 64 – Examination of the login tar file

[15] http://www.mynetwatchman.com/LID.asp?IID=100061132

As can be seen above in figure 64, by examining the 1login.tgz file the user and group information from the creator of the archive file is still intact. In this case the user is *sagi* and the group *root*.

The same analysis for the 1ssh2.tar.gz file results in the user and group information being obtained. In this case the user is mattanl and the group also mattanl:

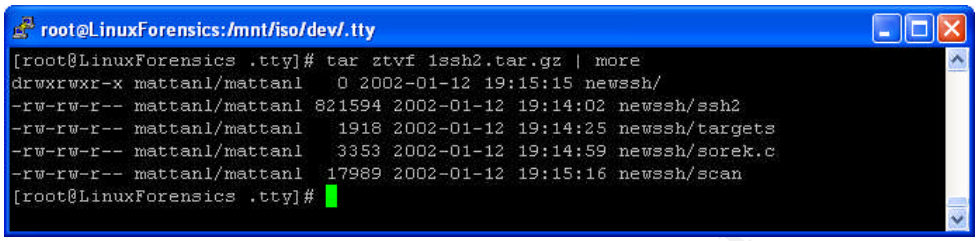


Figure 65 – Identification of the username mattanl

Search for the username sagi on Google returns thousans of hits, but a search for mattanl on Google resulted in only three hits. One of the postings is the following Internet posting concerning an attack on another system and also from a system in Israel:

http://seclists.org/lists/incidents/2001/Dec/0009.html

In addition to this, the username mattanl appears on a German hacker web site:

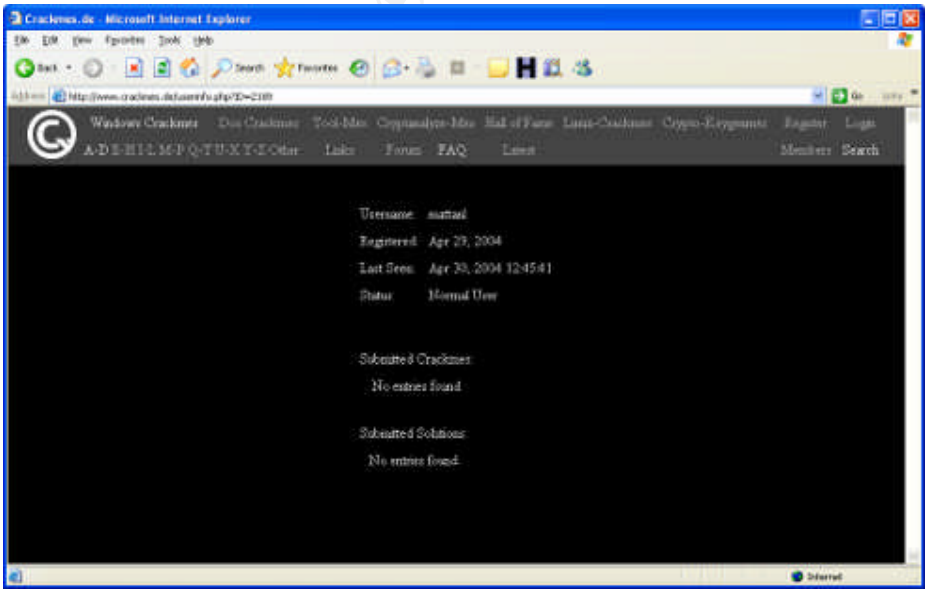


Figure 66 – Potential sighting of Mattanl on a German hacker web site

And, potentially most importantly of all, a personal website which may contain a mattanl's e-mail address and ICQ instant messaging details:

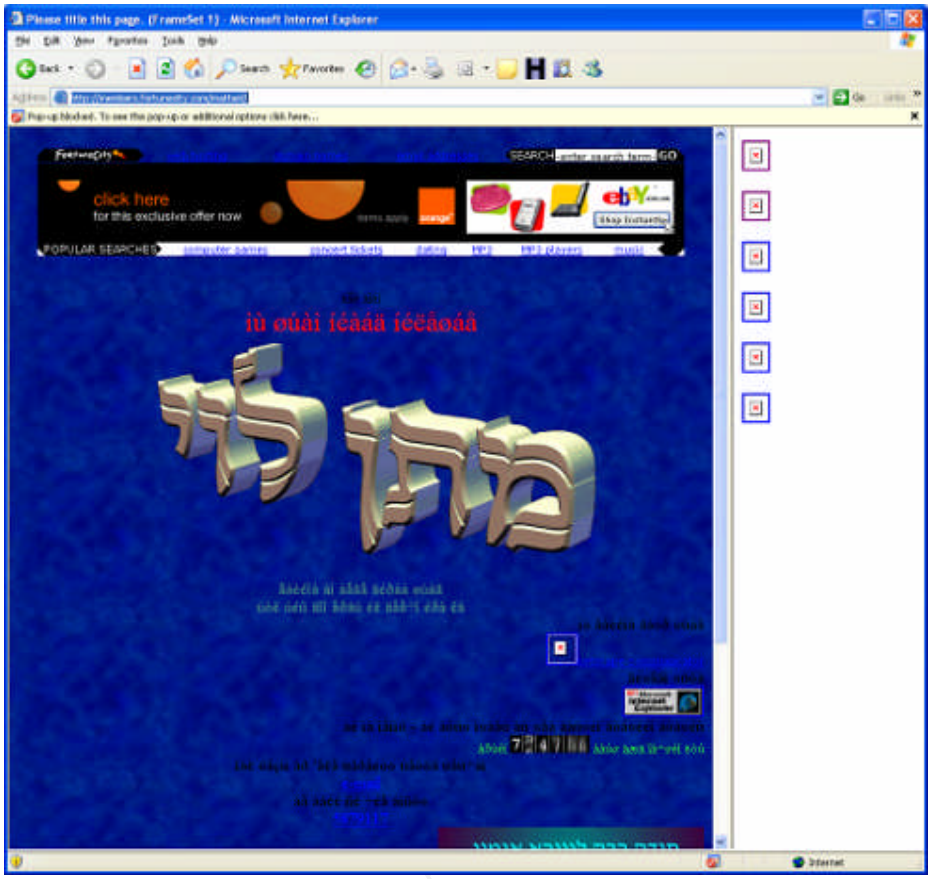


Figure 67 – Mattanl's personal web site?

The e-mail address shown is star10@inter.net.il which is the same ISP that the attack was launched from. The ICQ number is: 5879117 however the ICQ search for this user number reports that the user no longer exists.

It is also very useful to have a colleague who can read Hebrew, as the graphic on the website gave the surname Levi. This gives me the potential attacker name of Mattan Levi.

Backdoors or Trojans?

To check for SETUID or SETGUID files the following command was executed against the mounted disk image:

```
# find . \( -perm –004000 –o –perm –002000 \) –type f
```

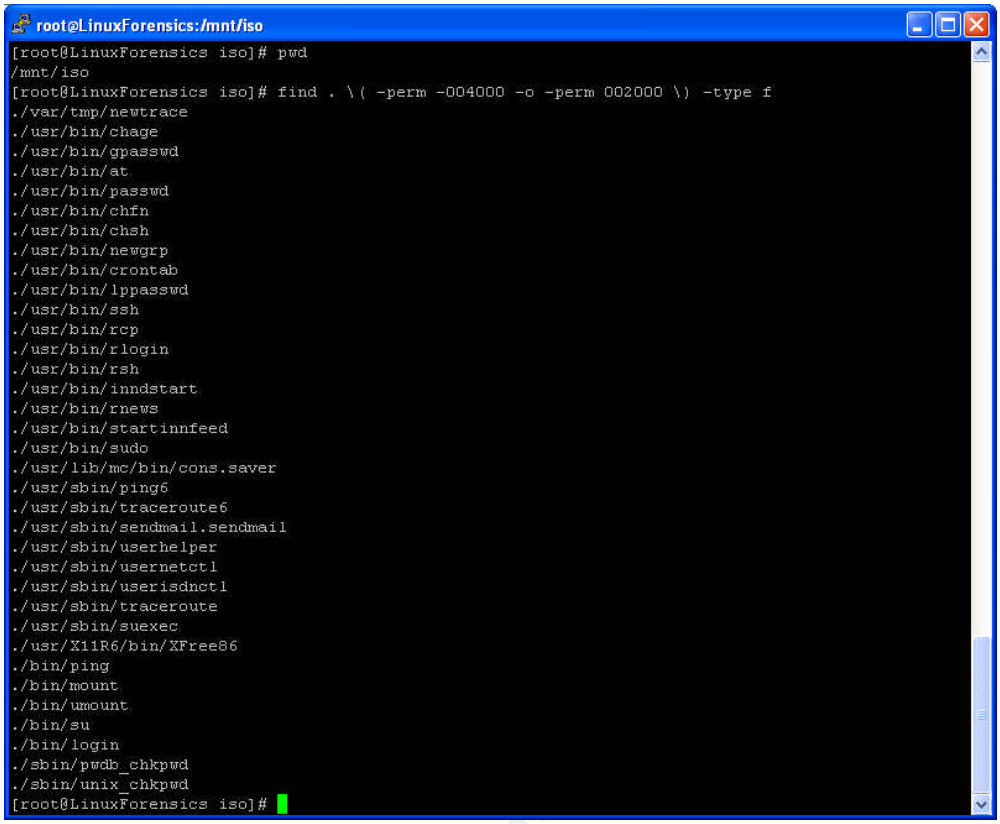


Figure 68 – Search for SETUID or SETGUID files

Only the /var/tmp/newtrace file was found to be unusual and I shall discuss this file in detail shortly. But first, I need to be able to manipulate the disk image in a forensically safe manner. I will now explain how this was done.

Mounting the honeypot disk image

For a closer look at the file system I can mount the image and navigate around the files as if I had the computer sitting in front of me.

Firstly, I need to understand what type of image I have:

```
[root@LinuxForensics evidence]# file honeypot.sda2.dd
honeypot.sda2.dd: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
```

Figure 69 – Using file to examine the disk image

As shown above, the disk image is a Linux revision 1.0 ext3 file system, which also needs to have the file system journal recovered. However, I do not want to allow my forensic workstation to allow the journal to be recovered, so I have to mount the image with ensuring data integrity as my primary concern.

The filesystem would normally be mounted using the command:

```
# mount –o loop filesystemname /mnt/iso
```

However, this will update the disk image as the journal would be recovered and the filesystemname would not be forensically sound. This can be seen in the example image below as the md5 signatures do not match.

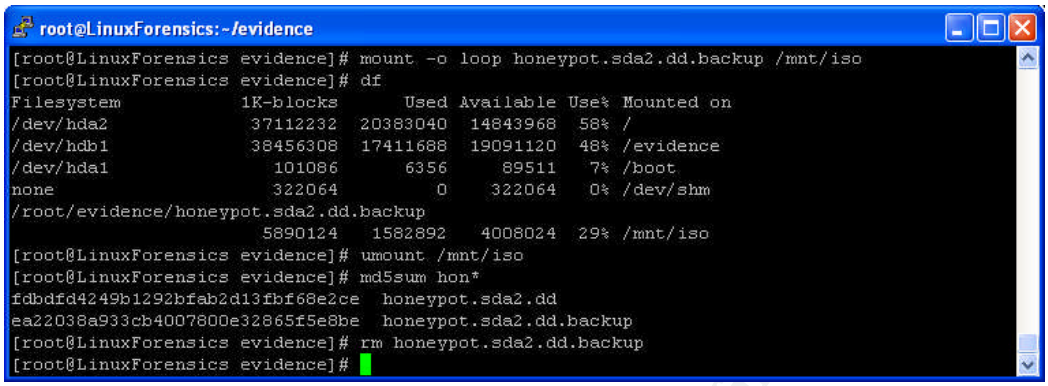


Figure 70 – mounting the ext3 filesystem (the wrong way)

To ensure that any further investigation on the ext3 filesystem is performed in a manner which will not alter the evidence, I installed a special modification to the Operating System which was designed by NASA[16].

The *enhanced_loopback* kernel was installed on the system, with the tools and modifications required. This allowed me to force the operating system into disallowing the ability to write any information to the disk.

To achieve this, I had to take a new forensic image of the compromised computer system, to include the whole disk rather than just the single filesystem I have analysed so far. As I showed earlier, the disk image was transferred to my forensic workstation and the MD5 checksums compared to ensure that it was a true image of the original.

[16] ftp://ftp.hq.nasa.gov/pub/ig/ccd/enhanced_loopback/

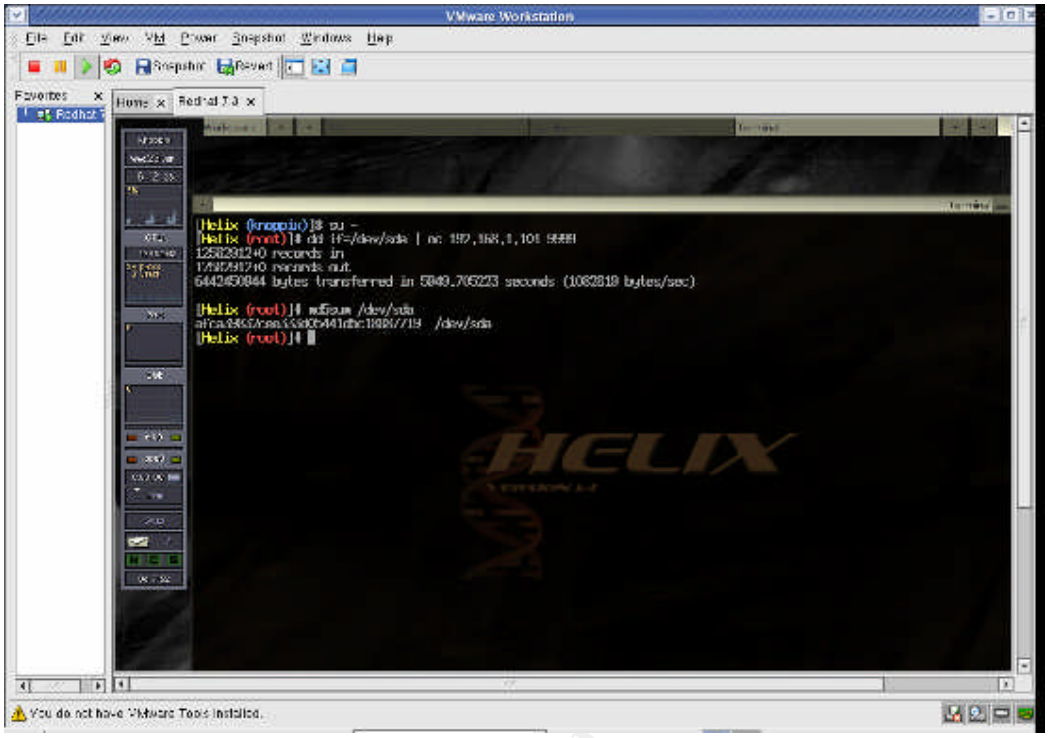


Figure 71 – Full disk image including MD5 checksum

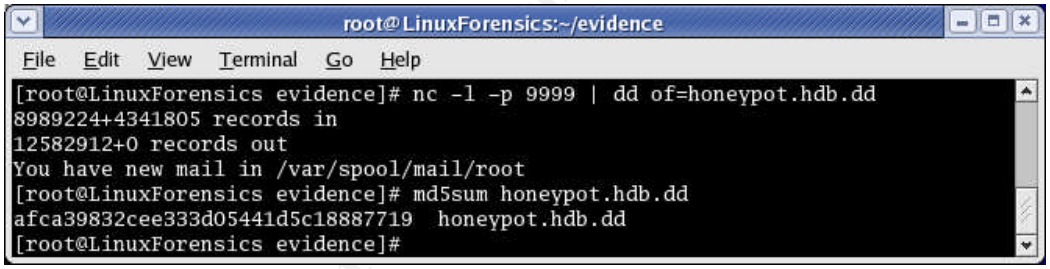


Figure 72 – Full disk image on Forensic Workstation including MD5 checksum

The new full disk image was then loaded using the new NASA commands to allow me to interact with it as if it were a physical disk rather than just an image. I shall describe the methods used to achieve this, and the results can be seen below.

The losetup command is used to assign read only access to the pseudo loop device which will control access to my disk image.

The sfdisk utility is used to show the layout of the disk image, and to identify that /dev/loopa2 is the partition on the disk I wish to analyse.

The disk is then mounted, again with the –ro (read only) option to ensure disk integrity.

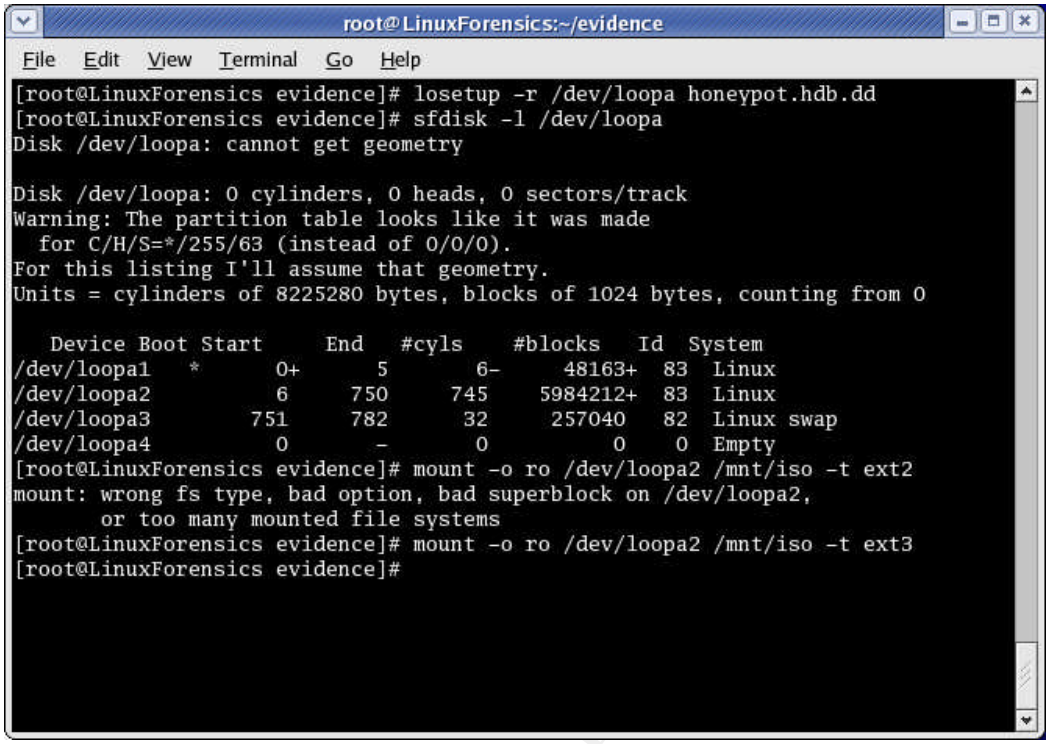


Figure 73 – enhanced_loop mounting of full disk image

Investigation into Newtrace

One of the first programs installed into the honeypot was /var/tmp/newtrace. I performed some initial analysis of the program as shown below:

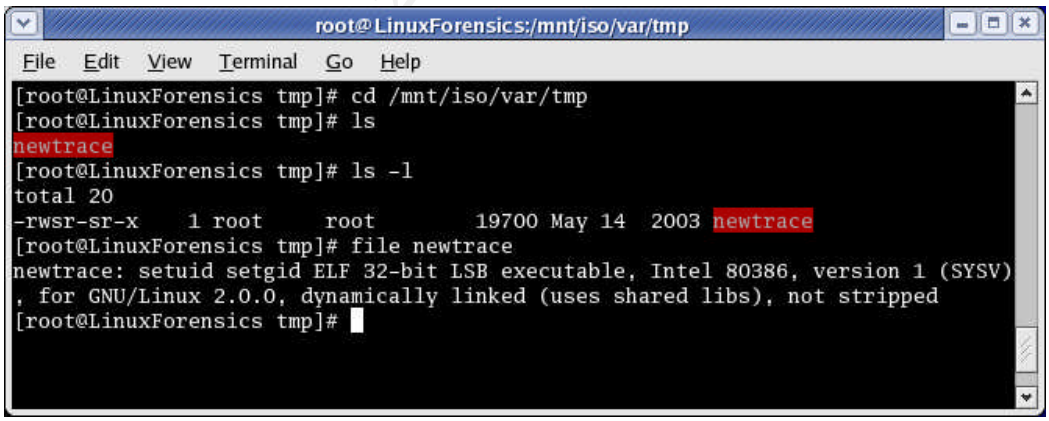


Figure 74 – Initial identification of /var/tmp/newtrace

As you can see from the above image, newtrace is highlighted in red. This indicates that special flags have been set on the file – in this case red does indicate danger. The flags are shown as:

-rwsr-sr-x

This indicates that SETUID and SETGUID have been set on the file forcing the file to execute with root user and root group permissions.

Performing a quick strings analysis of the newtrace binary may gave me enough information to check what the code is used for. The output displayed below has been truncated to only show key wording:

```
[-] Unable to read /proc/self/exe
[-] Unable to write shellcode
[+] Signal caught
[-] Unable to read registers
[+] Shellcode placed at 0x%08lx
[+] Now wait for suid shell...
[-] Unable to detach from victim
[-] Fatal error
[-] Unable to attach
[+] Attached to %d
[-] Unable to setup syscall trace
[+] Waiting for signal
[-] Unable to stat myself
root
/bin/sh
[-] Unable to spawn shell
[-] Unable to fork
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
GCC: (GNU) 2.95.3 20010315 (SuSE)
```

From the above text, newtrace would certainly appear to be a program which creates a new UNIX shell – highlighted in the /bin/sh line, and the error "Unable to spawn shell".

By searching for the phrase "Shellcode placed at 0x%08lx" I am able to identify a potential identification for the program. The source code for this program is available here from SecuriTeam.com[17]. The program is the ptrace exploit which gives the user elevated privileges on the system so that they can use *root* level commands. The can also see from the above strings output, that the code was compiled on a SuSE Linux system with the 2.95.3 GCC compiler.

Habitual Hacker or just a script kiddie?

It would appear from the evidence gathered, that the hacker who broke into the honeypot was a well organised individual and on a mission.

The hacker utilised two systems, one in Canada the other in Israel, to compromise the honeypot, and showed preplanning in the tools and specific intent on the use of the system he now had control of. The hacker also had access to a website hosted in the University of Gent, in Belgium to download his trojaned commands and utilities from.

[17] http://www.securiteam.com/exploits/5CP0Q0U9FY.html

The act of modifying the system to allow long term access, and removing the initial entry point are typical hacker tricks in ensuring no other hacker violates the captured system.

Part 3 – Legal Issues of Incident Handling

Based upon the type of material John Price was distributing, what if any, laws have been broken based upon the distribution?

The distribution of MP3 files in itself is not a criminal act. The distinction comes when the MP3 is a copy of a copyrighted original. Within the UK, the law that is broken is the Copyrights, Designs and Patents Act 1988[18]. Specifically, section 27(2) states:

"An article is an infringing copy if its making constituted an infringement of the copyright in the work in question".

It is clear from this statement that just the act of creating the MP3 from a source, such as a CD, where that original is under copyright is breaking the act. However, it becomes more complicated as under Section 107 of the same act you can commit an offence even if you didn't create the MP3, but:

Try to sell them, hire them, or other such distribution.

Section 107(1)[19]:

a) **makes for sale or hire, or**
b) **imports into the United Kingdom otherwise that for his private and domestic use, or**
c) **possesses in the course of a business with a view to committing any act infringing the copyright, or**
d) **in the course of a business**
    i. **sells or lets for hire, or**
    ii. **offers or exposes for sale or hire, or**
    iii. **exhibits in public, or**
    iv. **distributes, or**
e) **distributes otherwise than in the course of a business to such an extent as to affect prejudicially the owner of the copyright**

Even if you have a method of creating MP3's and you are suspected that you are trying to sell them, even if you are not found in possession of MP3's. Section 107(2):

a) **makes an article specially designed or adapted for making copies of a particular copyright work, or**

[18] http://www.legislation.hmso.gov.uk/acts/acts1988/Ukpga_19880048_en_1.htm

[19] http://www.legislation.hmso.gov.uk/acts/acts1988/Ukpga_19880048_en_7.htm#mdiv107

b) has such an article in his possession, knowing or having a reason to believe that it is to be used to make infringing copies for sale or hire or for use in the course of a business.

Or, by holding a public performance of the copied material. Section 107(3):

a) **by the public performance of a literary, dramatic or musical work, or**
b) **by the playing or showing in public of a sound recording or film, any person who caused the work to be so performed, played or shown is guilty of an offence if he knew or had reason to believe that copyright would be infringed.**

As it is assumed that John Price distributed known copyrighted material then he will have committed an offence under Section 107(1) of the Copyright, Design and Patents Act 1988. The act also gives guidance for the penalty for committing such an offence; however it does assume that the act was performed as part of a business. Although no direct evidence was found during the investigation that John Price was selling the MP3's he did have a screenshot from an e-bay site. If it was found that he was selling the MP3's as part of a business, then the act outlines in Section 107(4) that:

> "**On conviction on indictment the maximum penalty is 2 years imprisonment and/or an unlimited fine. On summary conviction the maximum penalty is 6 months imprisonment and/or a £5,000 fine**.[20]"

However, if the MP3's are not distributed as part of a business, then the act describes the following:

> "**All of the remaining offences under Section 107 are summary offences carrying a maximum penalty of 6 months imprisonment and/or a £5,000 fine**."

In addition to this, under sections 108 and sections 114 of the act, the court may order him to "deliver up" the copied goods, and other such articles which could include his computer systems and in turn these could be destroyed.

[20] http://www.fact-uk.org.uk/legislation%20fmset.htm

What would the appropriate steps be to taken if you discovered this information on your systems?

Computer misuse in a large organisation is of growing concern. However, how this is controlled and progressed through potential disciplinary procedures is often through old fashioned means.

Misuse of company systems, be they computer systems or anything else is often charged with 'Gross Misconduct' on the basis that the activities have been a waste of company time or may result in reputational damage.

This allows the process to be completed, and any disciplinary action taken without the spectre of proof being based on new and often misunderstood technologies.

However, the installation of non-approved software, or the use of a computer system "Beyond the Users Authority" are both disciplinary actions and could result in the employee being dismissed.

It may be worth considering amending the Acceptable User Policy issued to all staff to state that the use of copyrighted material on company systems is banned just to highlight that the company is taking the issue of John Price's actions seriously.

On discovery of such material in my place of work, the issue would be progressed to the Risk Manager of the business area involved. Once reported, both the Risk Manager and Group Internal Audit could be involved in the investigation.

Under the European Human Rights legislation, there is the potential impact of allowing "personal use" of company property – telephones, Internet Access, computer access for e-mail etc. The company may fall foul of these new laws and as yet these are untested in law.

In the event your corporate counsel decides to not pursue the matter any further at this point, what steps should you take to ensure any evidence you collect can be admissible in proceedings in the future should the situation change?

The passage of time has the potential of impacting the admissibility of evidence, therefore I need to consider what happens to the evidence I have collected, and received as part of this investigation.

The chain of custody procedures have to continue to be upheld even though the investigation, at this time, has ended. The handover of all collected, tagged and sealed evidence to a secure storage facility within the organisation also has to be entered on the evidence log.

Each item of evidence must be sealed in a manner that will detect any tampering, labelled evidence bags are suitable for this, and they should be tapped shut, and have signatures signed across the tape as seals.

The evidence from the forensic workstation should be imaged, md5 checksums taken, and this also entered into evidence.

All notes must be entered intact and complete, again these should be sealed as before.

How would your actions change if your investigation disclosed that John Price was distributing child pornography?

The issue of the discovery of child pornography during an investigation is an extremely serious one. There are a number of acts of parliament which would be broken should child pornography be found to be involved.

However, the situation is so delicate that it is advised that if such material is found during an investigation, that the investigation is stopped and the matter reported directly to the police. The Internet Watch Foundation[21] describe very clearly on their web site:

"Please note it is against the law to actively seek out such images and doing so in order to report to the IWF would not be a defence in court."

In this area of law, there are two potentially contradictory acts that could affect a forensic investigation. These acts are "The Protection of Children Act 1978" and "The Sexual Offences Act 2003". In addition to this, the remit of the Crown Prosecution service is to only bring prosecutions where it is deemed to be in the public interest. Therefore, it is possible that a forensic investigator may not be prosecuted under on this basis.

It is not clearly understood what would happen to a forensic investigator if they were to continue to perform an investigation once child pornography had been found. Therefore, it is safest to stop the investigation and hand the investigation over to the police.

The use of computer systems in the distribution of child pornography makes this situation more complicated. The Protection of Children Act 1978 under section 1 makes it clear what constitutes an offence under the act:

"To take, or permit to be taken, or to **make** any indecent photograph or pseudo-photograph of a child; or to distribute or show such indecent

[21] http://www.iwf.org.uk/

photographs or pseudo-photographs; or to possess such indecent photographs or pseudo-photographs, with a view to their being distributed or shown by himself or others; or to publish or cause to be published any advertisement likely to be understood as conveying that the advertiser distributes or shows such indecent photographs or pseudo-photographs, or intends to do so"

However, this was added to under Section 160 of The Criminal Justice Act of 1988, which made just the possession of such material a serious criminal act.

The use of the key phrase *pseudo-photograph* was included to cover such occasions where the evidence is a downloaded image, which the evidence may not consider an actual photograph.

The definition of "*make"* under the Protection of Children Act 1988, section 1 (as highlighted above) was redefined during R vs. Bowden (1999) to include the act of downloading images from the Internet, the storage of such images, or the printing of such images. This however, did not clarify the impact this would have on the provision of computer systems that conveyed the images. Therefore, a web cache could be seen as downloading or storing the image at which point the owners of the infrastructure becomes liable. This anomaly was cleared up in the new Communications Act 2003, in which the role of the service carrier was clarified.

The Internet Watch Foundation makes the following comment about the impact of the ability to download child pornography[22]:

"In Longmuir v. H.M.A. 2000 S.C.C.R. 447, it was upheld on appeal, that downloading images from the Internet was within Section 52(1) (a). The word "make" covered an activity whereby a computer was used to bring into existence data stored on a computer disk. A person who downloads images is making photographs. Operation of a computer to download electronic signals could be distinguished from mere possession of indecent photographs (where the possessor has not himself been responsible for bringing the material into existence)."

Finally, the most recent change concerning computer forensics can be seen in the Sexual Offences Act 2003 where – and to quote:

"In proceedings for an offence under section 1(1) (a) of making an indecent photograph or pseudo-photograph, the defendant is not guilty of the offence if he proves that it was necessary for him to make the photograph or pseudo-photograph for the purposes of the prevention, detection or investigation of crime, or for the purposes of criminal proceedings, in any part of the world"

I would not however, want to be the person who tests this change.

[22] http://www.iwf.org.uk/hotline/uk_law.html

References

Software used during the practical

Carrier, Brian. "The Autopsy Forensic Browser". URL: http://www.sleuthkit.org/autopsy/index.php (25th July 2004)

Carrier, Brian. "The Sleuth Kit". URL: http://www.sleuthkit.org/sleuthkit/index.php (25th July 2004)

Lord, Mark "mp3tool -- examine / modify .mp3 headers under Linux". URL: http://empeg-hijack.sourceforge.net/mp3tool.html (25th July 2004)

Provos, Niels "Steganography Detection with Stegdetect". URL: http://www.outguess.org/detection.php (25th July 2004)

Caswell, Brian and Roesch, Marty "Snort Intrusion Detection System". URL: http://www.snort.org (25th July 2004)

Luttgens, Jason. "Enhanced Loopback Linux Kernel" URL: ftp://ftp.hq.nasa.gov/pub/ig/ccd/enhanced_loopback/ (25th July 2004)

Further Information

Chuvakin, Anton "Linux Data Hiding and Recovery" 3rd October 2002. URL: http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html (25th July 2004)

Oliver, Paul "Wotsit's Format – The Programmers Resource" URL: http://www.wotsit.org/search.asp?page=5&s=music (25th July 2004)

Legal

Blyth, Andrew "Computer Misuse and Computer Law" URL: http://www.comp.glam.ac.uk/ism23/Additional-Material/Computer-Crime-&-Law.html (25th July 2004)

Federation against Copyright Theft, URL: http://www.fact-uk.org.uk (25th July 2004)

Her Magisties Stationary Office, URL: http://www.legislation.hmso.gov.uk (25th July 2004)

Copyright, Designs and Patents Act 1988, URL: http://www.legislation.hmso.gov.uk/acts/acts1988/Ukpga_19880048_en_1.htm (25th July 2004)

Internet Watch Foundation, URL: http://www.iwf.org.uk/ (25th July 2004)

http://www.cerias.purdue.edu/homes/carrier/forensics/docs/opensrc_legal.pdf

Kenneally, Erin E, "Gatekeeping Out Of The Box:
Open Source Software As A Mechanism To Assess Reliability For Digital Evidence" URL: http://www.vjolt.net/vol6/issue3/v6i3-a13-Kenneally.html (25th July 2004)

Honeypots

Honeynet Project "Know Your Enemy: Learning with VMWare". URL: http://www.honeynet.org/papers/vmware/ (25th July 2004)

Security Advisories, exploits, etc

CERT, "Advisory CA-2002-27 Apache/mod_ssl Worm",
URL:http://www.cert.org/advisories/CA-2002-27.html (25th July 2004)

Solar Eclipse, "Openssl-too-open remote exploit" URL:
http://packetstormsecurity.org/0209-exploits/openssl-too-open.tar.gz (25th July 2004)

Purczynski , Wojciech "Linux kernel ptrace/kmod local root exploit ", URL:
http://www.securiteam.com/exploits/5CP0Q0U9FY.html (25th July 2004)