



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>

Frank Adelstein

Submitted on: April 28, 2004

**An analysis of an unknown binary
and an evaluation of a tool to aid analyzing
the forensic footprint of archiving programs**

© SANS Institute 2004, Author retains full rights.

Abstract

This paper constitutes the practical examination for the GIAC Certified Forensic Examiner certification, version 1.4, and contains three parts. Part 1 presents an analysis of an unknown binary found on a floppy disk. Part 2 (option 2) evaluates a tool created to aid in the analysis of the forensic footprint of archiving programs. Part 3 discusses the legal issues involved in the case from Part 1.

© SANS Institute 2004, Author retains full rights.

Table of Contents

Abstract.....	2
Table of Contents.....	3
Table of Figures	4
Part 1 – Analysis of an unknown binary	5
Introduction.....	5
Summary	5
Initial steps	6
Binary details.....	6
Program description	8
Forensic details	21
Program identification.....	22
Legal implications.....	27
Interview questions.....	28
Case information	29
Additional information.....	32
Part 2 – Option 2: Forensic tool validation	34
Introduction.....	34
Scope	34
Tool description	35
Test apparatus	39
Environmental conditions	41
Description of the procedures.....	41
Criteria for approval.....	43
Data and results	44
Analysis.....	45
Presentation	45
Conclusions.....	47
Additional Information.....	48
Part 3 – Legal issues of incident handling.....	49
Question A.....	49
Question B.....	53
Question C	54
Question D	54
References	56
Appendix A – File timeline	58
Appendix B – Files sorted by inode.....	59
Appendix C – tar2d2 output from tar under Linux	60
Appendix D – tar2d2 output from tar under XP	63

Table of Figures

Figure 1 MD5 hash of unknown binary	8
Figure 2 Identification of unknown file type	10
Figure 3 string search for a false lead	11
Figure 4 run of prog with garbage options	11
Figure 5 prog run with --help	12
Figure 6 strace command to capture system calls	12
Figure 7 strace output of prog with --doc man	13
Figure 8 screen shot of strace of hello world	14
Figure 9 running prog on a file (without root privilege)	14
Figure 10 strace of prog with --mode s (without root privilege)	15
Figure 11 slack space extraction.....	16
Figure 12 contents of the hidden downloads file	16
Figure 13 rpm information on nc package.....	19
Figure 14 contents of file Mikemsg.doc.....	22
Figure 15 diff of bmap.c	24
Figure 16 diff of Makefile for bmap.....	25
Figure 17 comparison of hash of output of two binaries.....	27
Figure 18 screen shot of help screen with prog version and date.....	31
Figure 19 screen shot of config file on Windows XP Pro	36
Figure 20 tar2d2 config file for Linux.....	37
Figure 21 dynamic libraries used by perl under Linux.....	38
Figure 22 command to install perl XML package	39
Figure 23 Screen shot showing the version of tar used on Windows XP	40
Figure 24 read-only ISO image of test files.....	42
Figure 25 perl 1-liner to translate epoch time into a human-readable date	44
Figure 26 sample XML summary tags	46

Part 1 – Analysis of an unknown binary

Introduction

In this section, we¹ describe an analysis of an image of a floppy disk obtained from an employee (John Price) who was discovered by an audit to be using company computing resources to illegally distribute copyrighted material. In the scenario, his hard disk was wiped, and all that could be obtained was a floppy that was in the drive of his PC. Mr. Price denies the floppy belongs to him. The floppy contains files, including an unknown binary. The primary goal of this analysis is to determine the purpose of the binary and how it may have been used by Mr. Price.

The primary investigative machine is an IBM R32 series ThinkPad with 768Mb of memory and 40Gb of disk running Linux version Red Hat 9 [13]. It has a variety of forensic tools installed on it (they will be described as they are used), as well as VMware 4.0 [19], which allows “virtual machines” to run from within the environment. The main benefit of a virtual machine is the ability to restore it to a known state. Since this practical involves running an unknown binary which may compromise (or destroy) a system, the ability to rapidly undo all changes is highly desirable. Under VMware, we installed another Red Hat 9 system, which was used to run the unknown binary.

Note: In this report, any program names, file names, or quoted computer output will be shown in a fixed, courier font (`like this`) Output of files, when only a few lines long, will be quoted or shown in a screen shot. Longer output will be shown in a box with a grey background.

Summary

Before describing all of the detailed results of the analysis and the process that was performed, we will summarize the more important results we obtained. An analysis of the binary, combined with Internet based searches revealed the unknown binary to be a program called bmap, a tool to manipulate the slack space on files. Further, we found one file on the floppy disk had a file hidden in its slack space. The hidden file, once extracted and uncompressed, contained a list of sites to use to download illegal (copyrighted) MP3s. We describe the investigative process in detail, as well as other, lesser results obtained during the analysis.

¹ Note: All work was performed by the sole author of this paper; the use of the first person plural (i.e., “we”) in the text is merely a writing convention and sounds better to the author than writing “I” or continually using the term “the author.” And in general, reports of this type will be submitted by a collection of people representing a lab.

Initial steps

We downloaded the initial file, named `binary_v1_4.zip`, from the SANS web site. Running the program `unzip -l` lists the contents of the zip file. It showed 2 files: `fl-160703-jp1.dd.gz` and `fl-160703-jp1.dd.gz.md5`. The SANS web site lists the floppy image as `fl-160703-jp1.dd.gz`, so this makes sense. Next, we ran `md5sum` on the binary to compute the cryptographic hash of it, so we could verify it had been successfully downloaded without corruption. The hash is: `4b680767a2aed974cec5fbcbf74cc97a`. Printing the contents of the file `fl-160703-jp1.dd.gz.md5` by using the `cat` command produced an identical hash. In addition, the same MD5 hash was listed on the SANS web site. Therefore, the file transfer was successful.

The next step is to uncompress the gzip file and put it on a CD-ROM. The “.gz” suffix on a file indicates that it is compressed with the GNU zip command. Running the program `gunzip` on the compressed file `fl-160703.jp1.dd.gz`, produces the uncompressed file `fl-160703-jp1.dd`. We can verify that no data was lost during un-compression by running `gzip -c fl-160703-jp1.dd | md5sum` and verifying the hash is still `4b680767a2aed974cec5fbcbf74cc97a`.

We created a CD-ROM of the files `fl-160703-jp1.dd` and `fl-160703-jp1.dd.gz.md5`. By putting the data on a CD-ROM, we have essentially put a “write-block” on the file, so neither the contents of the files nor their metadata (e.g., time stamps) will change by any tools we use during the subsequent analysis. Finally, after creating the CD-ROM, we once again verified its hash via the command `gzip -c fl-160703-jp1.dd | md5sum`.

The `file` program identifies file types based on their content. Running `file` on `fl-160703.jp1.dd` indicates that it is a Linux ext2 file system image (Linux rev 1.0 ext2 filesystem data). We then mounted the CD-ROM on the Linux system using the “loop” device. This allows a file on a disk or CD containing an image of a file system to be interpreted as if it were a disk. It is mounted and the files on the file system are visible through the mount point. The mount command used was: `mount -o ro,loop,noexec,noatime /mnt/cdrom/fl-160703-jp1.dd /mnt/hack/unixforensics_mount/`. At this point, we could begin to look at the contents of the disk without fear of changing the files or the access time.

Binary details

The true name of the unknown binary named “prog” on the floppy is `bmap`, written by Daniel Ridge, email address: `newt@scyld.com`. The “Program description” and “Forensic details” sections describe how we discovered this.

The MAC time information (last modified, last accessed and last changed time) is:

Access: 2003-07-16 02:12:45.000000000 -0400

Modify: 2003-07-14 10:24:00.000000000 -0400

Change: 2003-07-16 02:05:33.000000000 -0400

The file's owner is ID number 502 and the group is ID number 502. The user and group names cannot be determined without the `/etc/passwd` and `/etc/group` files. However, it should be noted that Red Hat Linux systems begin assigning user IDs at 500, by default. This would imply that there were two other user accounts that had been created on the account in question, assuming the defaults were used. There is no way, however, to know if that was the case given the current evidence.

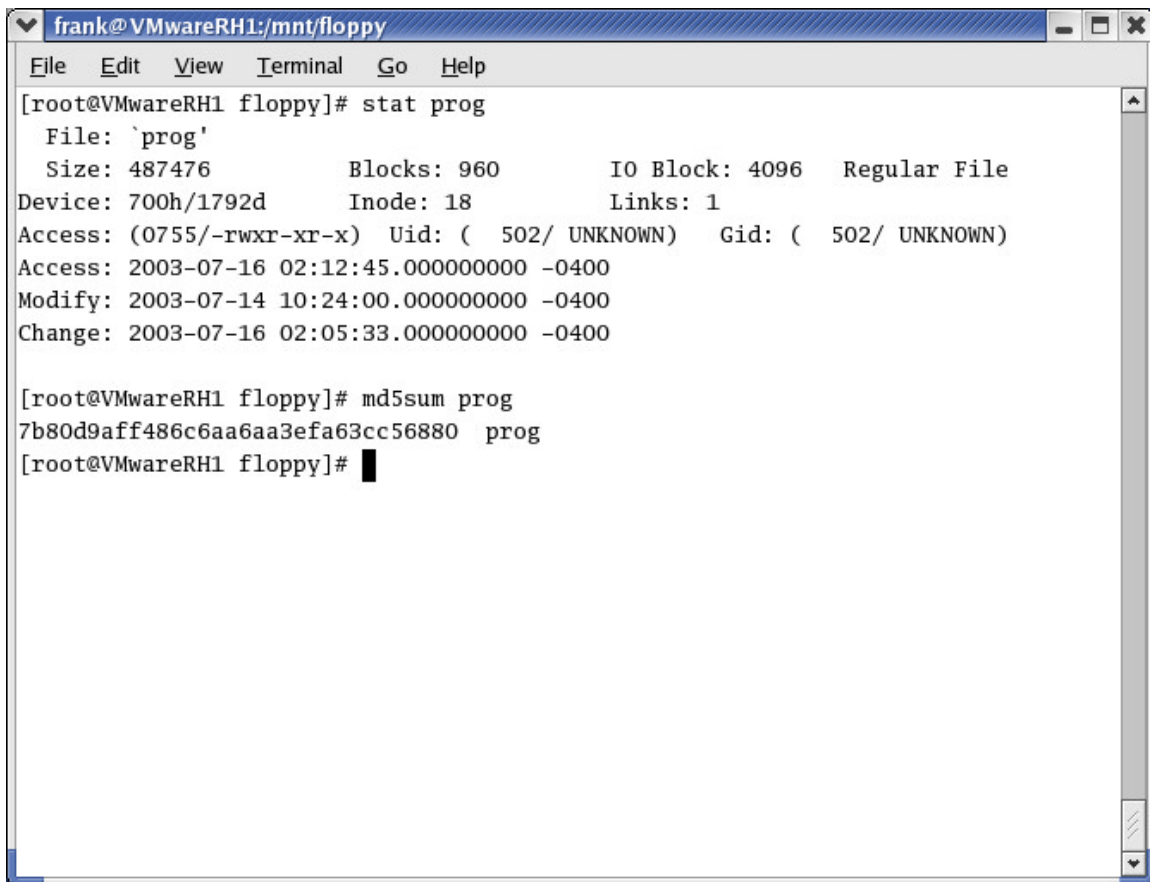
The file size of the binary called `prog` (in bytes) is 487476.

The MD5 hash of the binary called `prog` is:

7b80d9aff486c6aa6aa3efa63cc56880.

Figure 1 shows a screen snapshot with the file MAC times and MD5 hash, which was determined by using the programs `stat` and `md5sum`.

© SANS Institute 2004, Author retains full rights.

A screenshot of a terminal window titled 'frank@VMwareRH1:/mnt/floppy'. The terminal shows the output of the 'stat prog' command, which displays file metadata such as size (487476), blocks (960), IO block (4096), device (700h/1792d), inode (18), links (1), and permissions (-rwxr-xr-x). It also shows the output of the 'md5sum prog' command, which returns the MD5 hash '7b80d9aff486c6aa6aa3efa63cc56880' for the file 'prog'.

```
frank@VMwareRH1:/mnt/floppy
File Edit View Terminal Go Help
[root@VMwareRH1 floppy]# stat prog
  File: `prog'
  Size: 487476      Blocks: 960      IO Block: 4096   Regular File
Device: 700h/1792d  Inode: 18       Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 502/ UNKNOWN)   Gid: ( 502/ UNKNOWN)
Access: 2003-07-16 02:12:45.000000000 -0400
Modify: 2003-07-14 10:24:00.000000000 -0400
Change: 2003-07-16 02:05:33.000000000 -0400

[root@VMwareRH1 floppy]# md5sum prog
7b80d9aff486c6aa6aa3efa63cc56880  prog
[root@VMwareRH1 floppy]#
```

Figure 1 MD5 hash of unknown binary

We used the `strings` program to extract ASCII strings from the binary. The following is a list of the more distinctive strings that can be used as key words for subsequent searches to help identify the unknown binary.

venum, MFT_LOG_THRESH, mft_log_shutdown, checkfrag, newt, bmap_get_slack_block, bmap_get_block_count, bmap_raw_open, bmap_raw_close, bmap_get_block_size, bogowipe, newt, 1.0.20, 07/15/03, Brazil.

A false lead came from the strings “Keld Simonsen” and “keld@dkuug.dk” as these were found in the binary file, but are not part of the program. They are part of the standard libc library, which was included in this binary, which was statically compiled.

Program description

The program “prog” is “bmap,” a slack space tool. Broadly speaking, the program can be used to list available slack space in files, and store and retrieve data from the slack space in files. This becomes an effective tool for hiding information in an operating system. The slack space is the space left over on the last sector of the file. By default, disk blocks are 4K in Red Hat, so slack space

varies from 0 to 4095 bytes. Data stored in the slack space would not be listed in any `ls` command, nor would it show up in any `du` or `df` disk space usage listing. The slack data would also not be visible through any program examining the file containing the slack space, such as `grep` or `strings`. The only way to see the data is to look at the raw blocks.

In addition to storing and retrieving data from a file's slack space, `bmap` can wipe out data stored in the slack space as well as wiping out the entire file. Wiping ("bogowipe") is done by writing 0x00 (null), then writing 0xFF, then writing 0x00 to the slack space or file. `bmap` also can check if a file has data (i.e., non-null) stored in its slack space, print the number of bytes of slack space available, check if a file is fragmented (i.e., if the file spans non-contiguous sectors on the disk), and display any fragmentation points.

The last time "prog" was used was July 16th, 2003 at 02:12:45 EDT. This was the last time the file was accessed. It is very likely that this represents the last time it was run.

Description of the analysis process performed

First, we simply looked at the files present on the disk. The contents of the disk are shown below:

- Docs/
 - Letter.doc
 - Mikemsg.doc
 - DVD-Playing-HOWTO-html.tar
 - Kernel-HOWTO-html.tar.gz
 - MP3-HOWTO-html.tar.gz
 - Sound-HOWTO-html.tar.gz.
- John/
 - sect-num.gif
 - sectors.gif
- May03/
 - ebay300.jpg
- lost+found/
- prog
- nc-1.10-16.i386.rpm..rpm
- .~5456g.tmp

Running the `file` program on `prog` produced:

```
# file prog
prog: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), for GNU/Linux 2.2.5, statically linked, stripped
```

Figure 2 Identification of unknown file type

Since `prog` was a statically linked binary, it included all of the libraries it uses. And since it was stripped, function names would not be visible from a debugger.

Running `strings` on `prog` was inconclusive. `strings` suggested there was a built-in help option to display all of the command line parameters, however we could not learn anything useful because the actual help messages were generated dynamically. All that could be seen from the string extraction were disconnected fragments of help messages.

The help messages could be displayed in text, HTML, or `nroff` (“man page”) format. The messages were generated dynamically, depending on the help format requested. The text in the binary contained text fragments that were used to build the messages at run-time, so the actual command line parameters to control “prog” were not included in the (static) help text in the binary, just represented as “%s” (most likely, a `printf`-like call would replace this with the real parameter). This meant that we would need to run the binary in order to get a clear help message. This either suggested sophistication on the part of the perpetrator by making it difficult to determine the nature of the program without running it, or possibly dumb luck by picking a complex program to use. Later analysis, which showed the help messages were deliberately edited, suggests sophistication is more likely.

We used google [10] to search the web for various strings in the file, such as “bogowipe,” but that produced no hits. That meant that finding the source code would require more work—we would need to identify the program first and then find its code, rather than find the code from a string in the program. Searching for “newt” produced too many hits.

The name and email address strings present in the binary, Keld Simonsen and `keld@dkuug.dk`, yielded hits, but examination of his web pages suggest that he is a Linux contributor. And in fact, upon further examination, we determined that the string came from `libc` as shown in Figure 3, and is thus present in all Linux binaries.

```
[root@localhost tmp]# ll /lib/tls/libc-2.3.2.so
-rwxr-xr-x    1 root    root      1531064 Mar 13  2003 /lib/tls/libc-
2.3.2.so
[root@localhost tmp]# strings /lib/tls/libc-2.3.2.so | grep Keld
Keld Simonsen
C/o Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V
```

Figure 3 string search for a false lead

Because the help messages in the binary were split up, printed with %s conventions (which is how the C language printing library functions like printf() and fprintf() allow the user to specify a string variable), the only way to really get the usage help message would be to run `prog`.

After configuring the VMware Linux system, we created a non-root account called “frank” which was used to initially run `prog`. We ran the program via the `strace` command, so that all system calls would be recorded. That way, we could determine if the program was attempting to access restricted resources.

We made “frank” a non-root user in order to limit potential damage it could do. The file first needed to be on the disk, rather than CD-ROM to run, so we copied `prog` to the `/tmp` directory. We ran `prog` with garbage parameters to get a help message shown in Figure 4.

```
/tmp/prog -asdfzxcv
invalid options: -asdfzxcv
try '--help' for help.
```

Figure 4 run of prog with garbage options

The second time, we ran it with `--help` as a command line option. This produced the help message shown in Figure 5.

The help message is not all that helpful. It appears to deal with files in some way, and mentions troubling functions like “wipe”—but in general, we need to run more tests before we can make any determinations.

```

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 ...

```

Figure 5 prog run with --help

The next test was to run the program with the “--mode man” command line argument. We used the following command, shown in Figure 6, to run it under `strace` so we could generate a list of all of the system calls.

```

strace -o /tmp/prog.out.3 -tt /tmp/prog -doc man >
/tmp/prog.man

```

Figure 6 strace command to capture system calls

The `strace` output file (shown in Figure 7) did not appear to contain anything suspicious.

```

20:45:42.951344 execve("/tmp/prog", ["/tmp/prog", "--doc", "man"], [/*
29 vars */]) = 0
20:45:43.029135 fcntl64(0, F_GETFD)      = 0
20:45:43.031876 fcntl64(1, F_GETFD)      = 0
20:45:43.033575 fcntl64(2, F_GETFD)      = 0
20:45:43.034379 uname({sys="Linux", node="VMwareRH1", ...}) = 0
20:45:43.043918 geteuid32()      = 500
20:45:43.044687 getuid32()      = 500
20:45:43.045354 getegid32()     = 500
20:45:43.046014 getgid32()     = 500
20:45:43.047144 brk(0)          = 0x80bedec
20:45:43.048338 brk(0x80bee0c)   = 0x80bee0c
20:45:43.049098 brk(0x80bf000)   = 0x80bf000
20:45:43.049768 brk(0x80c0000)   = 0x80c0000
20:45:43.051319 fstat64(1, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
20:45:43.053797 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40000000
20:45:43.055717 write(1, ".TH PROG \"1\" \"07/15/03\" \"1.0.20 \"...,
1776) = 1776
20:45:43.064223 munmap(0x40000000, 4096) = 0
20:45:43.065294 _exit(0)          = ?

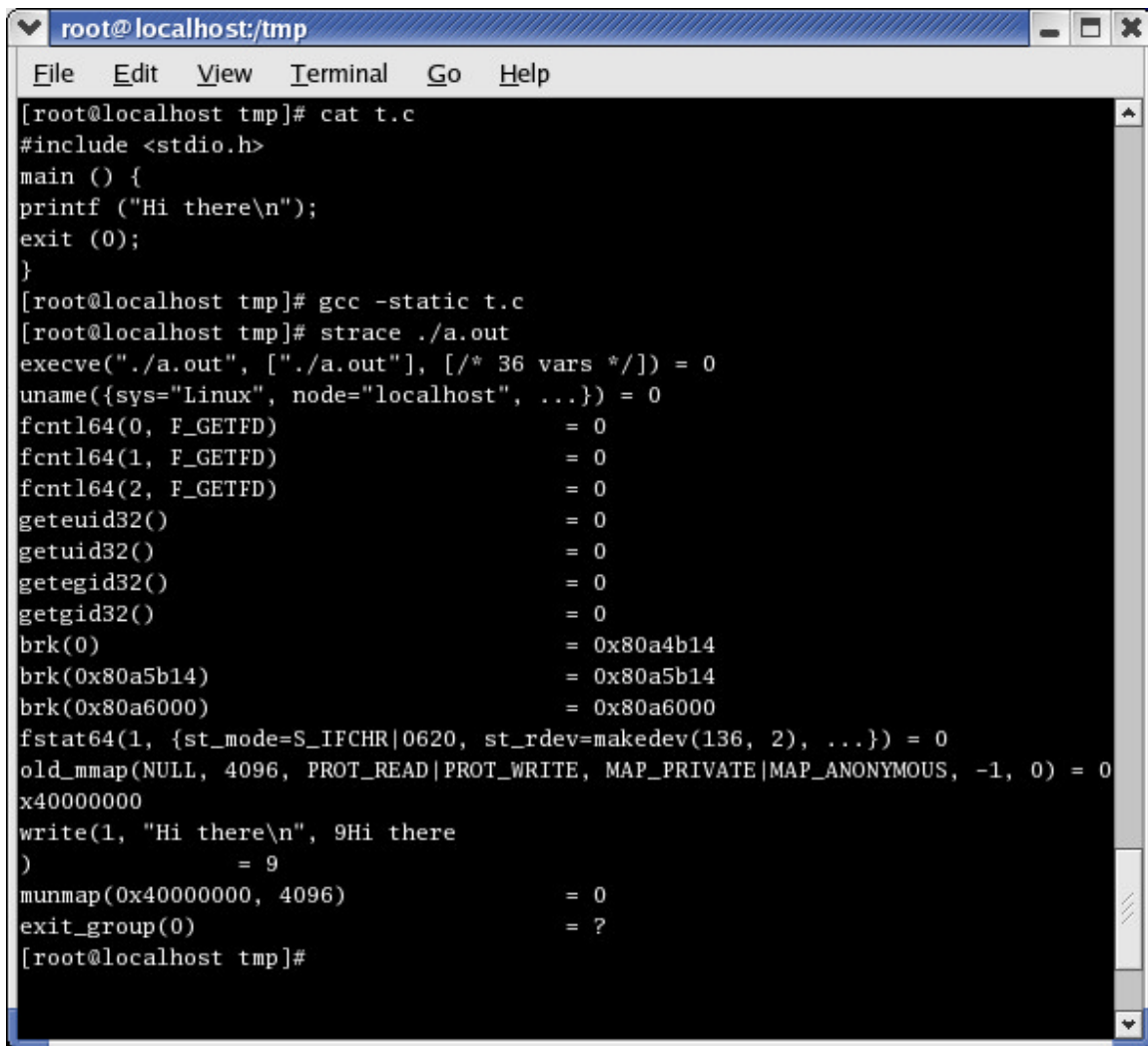
```

Figure 7 strace output of prog with --doc man

The `strace` output above shows the system calls the program made. Most of the code is standard entrance and exit code that comes from the run-time libraries linked in by the compiler. In order to determine what system calls make up the entrance code, we performed a simple test. We created a simple “hello world” style C program, compiled it statically, and then ran it under `strace` and compared the results to those shown in Figure 7.

The screen shot in Figure 8 shows the “hello world” style C program, followed by the static compiling of it and the run under `strace`. The output of `strace` produces the same calls, though not in the same order, as the `strace` output from `prog`. This allows us to determine what parts of `strace` are from the unknown binary and what is part of the normal C runtime library (`libc`).

By comparing the output from the two `strace` commands, it was apparent that all of the system calls shown in the `strace` of `prog` (Figure 7) were part of the entrance and exit code, *except* the third from the end call, i.e., “write.” This lets us know that, so far, the program is not doing anything hidden behind the scenes.



```
root@localhost:tmp
File Edit View Terminal Go Help
[root@localhost tmp]# cat t.c
#include <stdio.h>
main () {
printf ("Hi there\n");
exit (0);
}
[root@localhost tmp]# gcc -static t.c
[root@localhost tmp]# strace ./a.out
execve("./a.out", ["/a.out"], [/* 36 vars */]) = 0
uname({sys="Linux", node="localhost", ...}) = 0
fcntl64(0, F_GETFD) = 0
fcntl64(1, F_GETFD) = 0
fcntl64(2, F_GETFD) = 0
geteuid32() = 0
getuid32() = 0
getegid32() = 0
getgid32() = 0
brk(0) = 0x80a4b14
brk(0x80a5b14) = 0x80a5b14
brk(0x80a6000) = 0x80a6000
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40000000
write(1, "Hi there\n", 9Hi there
) = 9
munmap(0x40000000, 4096) = 0
exit_group(0) = ?
[root@localhost tmp]#
```

Figure 8 screen shot of strace of hello world

The help message suggested it was necessary to run `prog` against a file, shown in Figure 9 below.

```
strace -o /tmp/prog.out.5 -tt /tmp/prog -mode s John/sect-
num.gif
unable to open raw device /dev/loop0
unable to raw open John/sect-num.gif
```

Figure 9 running prog on a file (without root privilege)

The error indicated `prog` needed to be run as root. It was attempting to open the raw device (`/dev/loop0`) on which the specified file (`John/sect-num.gif`)

was mounted. The results of `strace`, shown in Figure 10, suggested `prog` was not doing anything unexpected.

```
20:51:37.671486 execve("/tmp/prog", ["/tmp/prog", "--mode", "s",  
"John/sect-num.gif"], [/* 29 vars */]) = 0  
20:51:37.761550 fcntl64(0, F_GETFD) = 0  
20:51:37.768467 fcntl64(1, F_GETFD) = 0  
20:51:37.769923 fcntl64(2, F_GETFD) = 0  
20:51:37.770797 uname({sys="Linux", node="VMwareRH1", ...}) = 0  
20:51:37.779011 geteuid32() = 500  
20:51:37.779757 getuid32() = 500  
20:51:37.780417 getegid32() = 500  
20:51:37.781287 getgid32() = 500  
20:51:37.782520 brk(0) = 0x80bedec  
20:51:37.787189 brk(0x80bee0c) = 0x80bee0c  
20:51:37.787927 brk(0x80bf000) = 0x80bf000  
20:51:37.788601 brk(0x80c0000) = 0x80c0000  
20:51:37.790166 lstat64("John/sect-num.gif", {st_mode=S_IFREG|0755,  
st_size=19088, ...}) = 0  
20:51:37.796685 open("John/sect-num.gif", O_RDONLY|O_LARGEFILE) = 3  
20:51:37.798355 ioctl(3, FIGETBSZ, 0xbffff214) = 0  
20:51:37.799654 lstat64("John/sect-num.gif", {st_mode=S_IFREG|0755,  
st_size=19088, ...}) = 0  
20:51:37.802134 lstat64("/dev/loop0", {st_mode=S_IFBLK|0660,  
st_rdev=makedev(7, 0), ...}) = 0  
20:51:37.818124 open("/dev/loop0", O_RDONLY|O_LARGEFILE) = -1 EACCES  
(Permission denied)  
20:51:37.820294 write(2, "unable to open raw device /dev/l...", 37) =  
37  
20:51:37.822899 write(2, "unable to raw open John/sect-num...", 37) =  
37  
20:51:38.000240 _exit(6) = ?
```

Figure 10 `strace` of `prog` with `--mode s` (without root privilege)

From this trace, we can see that the program first opens the specified file (`sect-num.gif`) in read-only mode. It “stats” the file to get information on it, which includes the physical device where the file resides—in this case `/dev/loop0`. The program then stats that file and then tries to open the raw device, again in read-only mode. It then prints the error messages we saw. To find out more about the program, we needed to run it as root.

We tried to minimize the risks of running an unknown binary on our machine by examining what the program was attempting to do. Our final protection was that we were using a virtual machine, with a saved snapshot and a back up as well. In the event of a disaster, we could easily restore the machine to its previous state

Repeating the same command as root showed that `prog` was looking at the inodes of the files. Because `prog` was trying to open the raw device, going

below the file system level, we formulated a working theory—this might be a program to hide data in the slack space of files.

A search on google for the terms “linux slack space” yielded a web page [5] by Anton Chuvakin, which says “the obscure tool bmap exists to jam data in slack space, take it out and also wipe the slack space if needed.” The text printed from the examples in the page suggested that the unknown binary is “bmap.” The web page gave a reference to an FTP site that had the source for the tool [2].²

We downloaded the program, `bmap`, and noted that the most recent version was 1.0.20, which matched one of the strings in `prog`. We compiled `bmap` by running `make`. We then ran `bmap` and observed its output, which was very similar to the output of `prog`, except the command names were full words indicative of their function.

After compiling `bmap`, we ran it on files in the `John` and `Doc` directories. Most yielded nulls in the slack space. However, the file `Docs/Sound-HOWTO-HTML.tar.gz` yielded a 185 byte file. It was the only file on the disk that contained data in the slack space. The `file` program identified the file that was extracted from the slack space as a compressed file named “downloads.”

```
/tmp/prog --mode s Docs/Sound-HOWTO-HTML.tar.gz | file -  
getting from block 190  
file size was; 26843  
slack size: 805  
block size: 2024  
standard input:  gzip compressed data, was "downloads", from Unix
```

Figure 11 slack space extraction

We decompressed this file and displayed its contents by: `prog --mode s Sounds-HOWTO.html.tar.gz | gunzip -c`. The contents of the file are shown in Figure 12. Bingo!

```
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***
```

Figure 12 contents of the hidden downloads file

² Note that since this was first written, the source is no longer available via the Scyld FTP site. It is still available at other mirror sites [3].

We take an MD5 hash of the file since it will be evidence. The MD5 hash of the file `downloads` is: `fb13acabc77f1d562fd7397cd7b230af`.

We had extracted a hidden file from the floppy. The next steps involved a detailed study of the program and an examination of the other files on the disk for any supplementary clues.

GIF files

There are two files in the John directory, `sect-num.gif` and `sectors.gif` both of which were confirmed as being GIF image data, version 87a, by the `file` program. We did a google image search for “sect-num.gif” and only one hit came up, at: http://www.cf-intl.com/evidence_recovery_basics.htm. We downloaded the image and took an MD5 hash of it, which was:

`95246c5b7112efd7dff3ae9aeac22f24`. However, the MD5 hash of `John/sect-num.gif` was: `636be3f63d098684b23965390cea0705`. In addition, the file sizes differ, with `John/sect-num.gif` containing 9088 bytes and the other 17640. Under the `xview` image viewing program, however, they appear identical.

On the same web page is a file `eviden3.gif` which corresponds to the image `John/sectors.gif`. Again, the file sizes differ (13793 and 20680, respectively), and this time the images look similar but slightly different (the font on the `John/sectors.gif` file looks thinner), but the images are very similar. It is possible that something else is embedded in the images.

Depending on available time and goals, we could use steganography tools to analyze the images. Our goal, however, was simply to identify the unknown binary. The content of the file named `downloads` suggests we should look for MP3 files, but such files would be too big to fit in the 7-8 Kbytes in these images.

netcat

We concluded that the file `nc-1.10.16.i386.rpm..rpm` on the floppy is the standard `nc` (netcat) program in an RPM (Red Hat Package Manager) format, since we were able to find the identical file at an archive site. We determined that as follows:

The `file` program identified the file `nc-1.10.16.i386.rpm..rpm` as RPM file (RPM v3 bin i386 nc-1.10-16). We did a search on google [10] for “nc-1.10.16.i386.rpm” in case the extra “..rpm” at the end was extraneous. We found several sites listing it as the rpm package for `nc` for Red Hat 8.0. We downloaded the file `nc-1.10.16-i386.rpm` from a mirror site [11] and then compared the downloaded file to the one from the floppy image. Both files were 56950 bytes. `md5sum` showed that both files had the same hash of:

535003964e861aad97ed28b56fe67720. Therefore, we can be somewhat confident that this is the standard nc (netcat) program. netcat is used to send and receive data across a network.

Since the file is an RPM (Red Hat package manager), the nc binary itself would most likely be installed in the directory specified in the RPM. Running RPM with the `-qlip` flags shows it installs files in `/usr/bin/nc`, as well as `/usr/share/man` and `/usr/doc` by default (see Figure 13). These files would be on the hard disk, rather than the floppy disk. The last MAC time was the c-time, possibly indicating the file was renamed, moved, or touched. Based on the large number of c-times late in the timeline (shown in Appendix A), it seems likely that the latter option, touch, is the most likely cause of the time stamp.

© SANS Institute 2004, Author retains full rights.

```
# rpm -qlip nc-1.10-16.i386.rpm..rpm
Name       : nc                               Relocations: (not
relocateable)
Version    : 1.10                           Vendor: Red Hat, Inc.
Release    : 16                             Build Date: Tue 23 Jul 2002
12:47:55 PM EDT
Install Date: (not installed)                Build Host: astest
Group      : Applications/Internet           Source RPM: nc-1.10-
16.src.rpm
Size       : 114474                          License: GPL
Signature  : DSA/SHA1, Tue 03 Sep 2002 05:30:55 PM EDT, Key ID
219180cddb42a60e
Packager   : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Summary    : Reads and writes data across network connections using
TCP or UDP.
Description :
The nc package contains Netcat (the program is actually nc), a simple
utility for reading and writing data across network connections, using
the TCP or UDP protocols. Netcat is intended to be a reliable back-end
tool which can be used directly or driven by other programs and
scripts. Netcat is also a feature-rich network debugging and
exploration tool, since it can create many different connections and
has many built-in capabilities.
/usr/bin/nc
/usr/share/doc/nc-1.10
/usr/share/doc/nc-1.10/Changelog
/usr/share/doc/nc-1.10/README
/usr/share/doc/nc-1.10/scripts
/usr/share/doc/nc-1.10/scripts/README
/usr/share/doc/nc-1.10/scripts/alta
/usr/share/doc/nc-1.10/scripts/bsh
/usr/share/doc/nc-1.10/scripts/dist.sh
/usr/share/doc/nc-1.10/scripts/irc
/usr/share/doc/nc-1.10/scripts/iscan
/usr/share/doc/nc-1.10/scripts/ncp
/usr/share/doc/nc-1.10/scripts/probe
/usr/share/doc/nc-1.10/scripts/web
/usr/share/doc/nc-1.10/scripts/webproxy
/usr/share/doc/nc-1.10/scripts/webrelay
/usr/share/doc/nc-1.10/scripts/websearch
/usr/share/man/man1/nc.1.gz
```

Figure 13 rpm information on nc package

Step-by-step analysis of the bmap tool

We have identified the program named `prog` on the floppy as the program `bmap`. We obtained the source to `bmap` on the Internet. The following analysis is based on the two source files in the `bmap 1.0.20` distribution, `bmap.c` and `libbmap.c`. Output from `strace` confirms that program flow matches the source listing. To limit space, we are presenting a “walkthrough” of the `main()` function and summarize elsewhere the purpose of the ancillary function calls.

After initializing variables and the logger (by calling `mft_log_init()`), it parses the command line using `mft_getopt()`, a variant of the `getopt()` function provided with the source, and performs some simple error checking to ensure that a filename is provided. The string “try --help for help” is printed if there is an argument error.

It then performs an `lstat()` of the filename and verifies that the file is a regular file and not a link (and prints an error message and exits if that is not the case). If there is more than one link to the file, it prints a warning message. Then it sets the output to be the output filename, if specified, otherwise it uses `STDOUT`.

Then it opens the file read-only. This is done in order to determine the device upon which the file is mounted. Opening a file does *not* modify the access time on a file; only performing the actual file read will change the access time. `bmap` does *not* read the file through this mechanism and thus leaves no “fingerprint” from this operation. (This was verified with a simple program that `open()`'ed a file then closed it, under Red Hat 9, and then comparing the MAC times reported by the `stat` program before and after running the test.)

Next, the program obtains the block size of the file via the `bmap_get_block_size()` function call (which is essentially a wrapper around the `FIGETBSZ ioctl()` call, which gets the block size for a file).

Then, it determines if the raw device must be opened as read or write only. If the function to be performed is “carve,” “wipe,” or “putslack” then the mode is write, otherwise it is read. And then the raw device is opened via the `bmap_raw_open()` function.

Next, the block count is determined by calling `bmap_get_block_count()`. Then it allocates a memory buffer (unless the operation is “map”). It stores the file size and checks if the file is sparse, i.e., the file size is greater than the block size multiplied by the number of blocks. If so, it prints a message indicating the file is sparse (“has holes in it”).

Then, the program iterates through all the blocks in the file. It does this by calling `bmap_map_block()` which is a wrapper around the `FIBMAP ioctl()`, which returns a block number on the physical device that corresponds to a point in the file. If the program was run in “carve” mode, it then seeks that block in the raw device, reads the block and writes it to output. If it was run in “wipe” mode, it calls `bogowipe()` on the block, which then writes `0x00`, then `0xFF`, followed by `0x00` to the block. If it was run in “frag” mode, it prints where the file is fragmented. A file is fragmented when it has non-contiguous blocks, which can be determined by checking if the current block is not the immediate successor (i.e., 1 greater than) the previous block. If it was run in “checkfrag” mode, then it simply sets a flag to true if fragmentation is detected. If run in “map” mode then it prints information about each block. All modes described in this paragraph must iterate over all the blocks in the file.

The final three modes only operate on the final block of the file. After completing the iteration, the program calculates an offset to the end of the data in the file. It determines the number of bytes available in slack space by subtracting the file size from the number of blocks multiplied by the block size.

If the program was run in “slack” mode, it seeks to the end of the file and then reads and writes the remaining data in the block.

If the program was run in “putslack” mode, it seeks to the end of the file, reads data in from standard input, and writes it into the slack space of the file.

If the program was run in “wipeslack” mode, then it calls `bogowipe()` on the slack space, which writes 0x00, then 0xFF and then 0x00 to the slack space (the remainder of the last block).

If the program was run in “checkslack” mode, then it reads in the slack space and checks if any data there is non-zero and returns true if so, false otherwise.

If the program was run in “slackbytes” mode, then it prints the number of bytes available in the slack space.

Finally, the program closes the raw device and target file, and the output file, if not `STDOUT`, and frees memory, if it was allocated. If the mode was “checkslack” or “checkfrag” the program will print a message saying whether there was or was not slack space or fragmentation, as appropriate.

Forensic details

In this section, we discuss the forensic footprints left by the program. The program stores data in the slack space of files, within the last block of the file, which is up to 4096 bytes long. No formal installation is required with `bmap`, so the presence of the binary is the only forensic footprint resulting from its “installation.”

`bmap` opens the raw block device which contains the file system that contains the file specified as a command line argument.

`bmap` stores data in the slack space of files by writing directly to the filesystem by using the raw device. Unfortunately (or remarkably), opening and closing a raw device this way does *not* affect the MAC times of the raw device. The tests we performed showed that, other than the presence of (non-null) data stored in the slack space itself and the access time on the binary, the program leaves no other traces that it had been run.

The program is statically compiled, so it does not use any shared libraries. Analyses of the program using `strace` as well as the source code showed that the only files the program manipulates are the target file and the raw device on which the target file exists.

Our analysis obtained a number of leads for further investigation. We extracted the file named `downloads` from the slack space of `Sounds-HOWTO.html.tar.gz`. The contents of the file are shown in Figure 12. Obviously, these URLs are leads that should be investigated. Note: none of these sites actually exist, since this is not a real case. We verified that using the `whois` command to look up these domain names.

In addition, there were two Word `.doc` files in the Docs directory. The file `Mikemsg.doc` contains the text (obtained by using Microsoft Word 2002 version 10.0):

Hey Mike,

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

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

JP

.

Figure 14 contents of file Mikemsg.doc

The properties in the file show “John Price” as the author. The MAC times of the `Mikemsg.doc` file all are Monday, July 14 2003 at 10:48:15am.

The file `Letter.doc` was a blank template for a letter and also listed “John Price” as the author in the properties. It may indicate he was sending a mass mailing out for distribution.

In the binary itself, there was the version number, 1.0.20, the compile date of 07/15/03, and the name “newt.” But “newt” points to the author of the tool, not to what it was used for or who used it.

Program identification

The source code for `bmap` is available through anonymous FTP at: ftp://ftp.scyld.com/pub/forensic_computing/bmap/.

Note: as of April 2004, this FTP directory is no longer available. `bmap` 1.0.20 is available through a mirror site at: <http://ftp.cfu.net/mirrors/garchiv.cs.uni.edu/garchiv/bmap-1.0.20/bmap-1.0.20.tar.gz>.

Earlier, we described how we found the `bmap` program and located the source on the Internet. Below, we described the evidence we gathered to demonstrate that the program named `prog` on the floppy disk was `bmap` version 1.0.20.

We analyzed the `bmap` program and compared its output to the unidentified program. The lists of strings extracted from the two programs have a great deal of overlap, although they are not identical. Both include all of the search terms mentioned above, such as “bogowipe”, “MFT_LOG_THRESH”, “newt”, and the various `bmap` strings. The next step was to attempt to make trivial changes to `bmap` that would get the two programs to match; however we realized that it was unlikely that we would get an exact binary match (verified with MD5 hashes), since the entrance code was *different* in the two libraries. We decided to see how closely we could match the programs.

The command line arguments were different in the two programs, to obscure the meaning of `prog`. So, we made minor changes to 8 lines in the `c` program `bmap.c`. These changes are shown in Figure 15.


```
root@localhost:tmp/bmap-1.0.20
File Edit View Terminal Go Help
[root@localhost bmap-1.0.20]#
[root@localhost bmap-1.0.20]#
[root@localhost bmap-1.0.20]# diff bmap.c*
64c64
<             {"m","list sector numbers",
---
>             {"map","list sector numbers",
66c66
<             {"c","extract a copy from the raw device",
---
>             {"carve","extract a copy from the raw device",
68c68
<             {"s","display data",
---
>             {"slack","display data in slack space",
70c70
<             {"p","place data",
---
>             {"putslack","place data into slack",
72c72
<             {"w","wipe",
---
>             {"wipeslack","wipe slack",
74c74
<             {"chk","test (returns 0 if exist)",
---
>             {"checkslack","test for slack (returns 0 if file has sla
ck)",
76c76
<             {"sb","print number of bytes available",0,MO_INT_CAST(BM
AP_SLACKBYTES)},
---
>             {"slackbytes","print number of slack bytes available",0,
MO_INT_CAST(BMAP_SLACKBYTES)},
104c104
<         "prog",
---
>         "bmap",
[root@localhost bmap-1.0.20]#
```

Figure 15 diff of bmap.c

In addition, we made three changes to the Makefile to set the compile date, change the author to "newt" (deleting his email address), and to compile the program statically. These changes are shown in Figure 16, in addition to a directory listing (via the `ls` command) to show all the files in the bmap top level source directory.

```
root@localhost:tmp/bmap-1.0.20
File Edit View Terminal Go Help
< {"w","wipe",
---
> {"wipeslack","wipe slack",
74c74
< {"chk","test (returns 0 if exist)",
---
> {"checkslack","test for slack (returns 0 if file has sla
ck)",
76c76
< {"sb","print number of bytes available",0,MO_INT_CAST(BM
AP_SLACKBYTES)},
---
> {"slackbytes","print number of slack bytes available",0,
MO_INT_CAST(BMAP_SLACKBYTES)},
104c104
< "prog",
---
> "bmap",
[root@localhost bmap-1.0.20]# diff Makefile*
7,10c7,8
< #BUILD_DATE = $(shell date +%D)
< BUILD_DATE = "07/15/03"
< #AUTHOR = "newt@scyld.com"
< AUTHOR = "newt"
---
> BUILD_DATE = $(shell date +%D)
> AUTHOR = "newt@scyld.com"
30c28
< CFLAGS = -Wall -static
---
> CFLAGS = -Wall -g
[root@localhost bmap-1.0.20]# ls
bclump.c      bmap.spec    dev_entries.c  LICENSE      slacker.c
bmap          bmap.tex     dev_entries.o  Makefile     slacker-modules.c
bmap.c        config.h     include        Makefile.orig
bmap.c.orig   COPYING     index.html     man
bmap.o        dev_builder  libbmap.c      mft
bmap.sgml.m4 dev_builder.c libbmap.o      README
[root@localhost bmap-1.0.20]#
```

Figure 16 diff of Makefile for bmap

After compiling a static version of `bmap`, we stripped the program of the symbol table (via the `strip` command). This version of `bmap` was able to read slack space stored on the evidence disk.

However, as we suspected, the MD5 checksum of our version did not match the MD5 checksum of `prog`. The problem is that we do not know on what system the binary was compiled. Even within the same version of Red Hat (e.g., 9.0), libraries differ. After compiling it on a Red Hat 9.0, we tried it on an 8.0 system,

but neither the file size nor MD5 hash matched the MD5 hash of `prog`. The file size was 543908 and the hash was f67dec94a73c3effda9fb47cf71a693d for RH9, and 526576 and 7d3f4f999857aff301c343df5e98b1db for RH8.

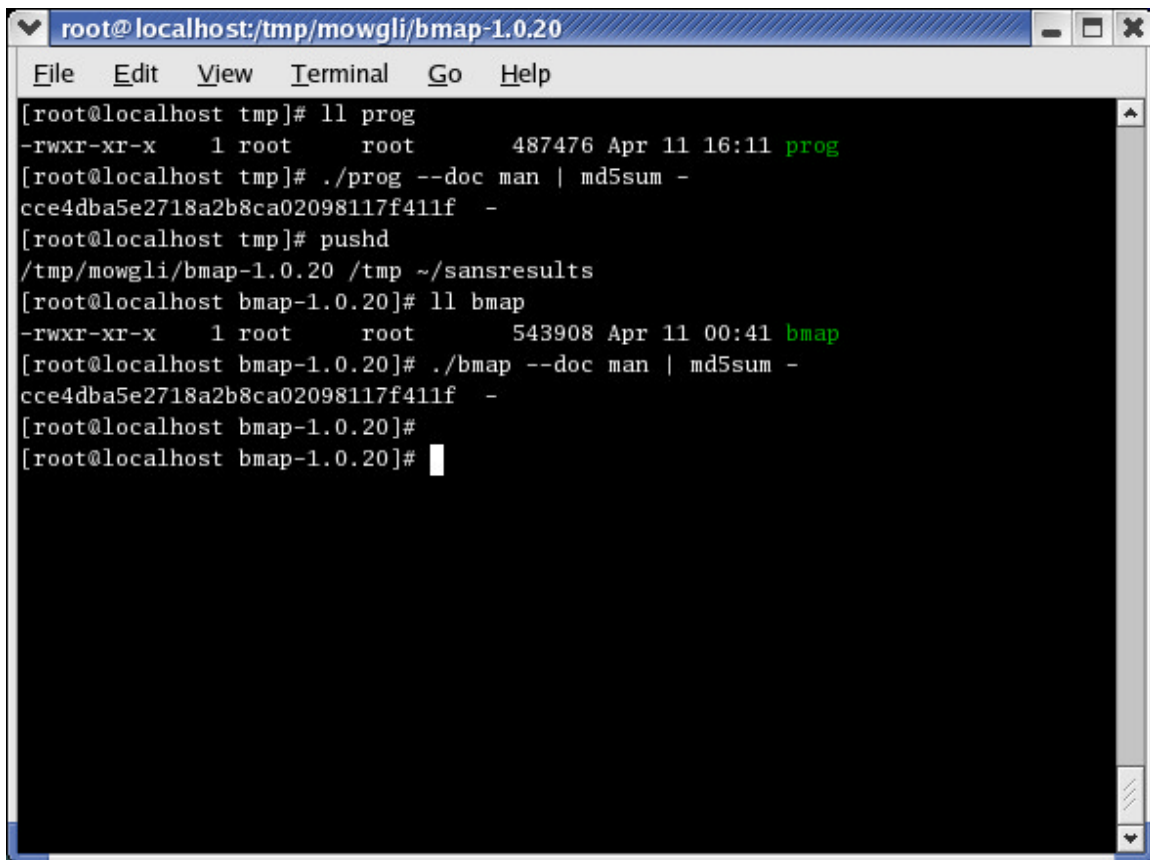
Failing to show the hashes matched, and not having ready access to all versions of Linux distributions, including Red Hat, Debian, SuSE, Slackware, etc. and the shared libraries, we tried the next best thing.

We generated output by running both programs in help mode (with the command line parameters `--doc man`, `--doc sgml`, and `--help`). We compared the output in 3 ways—`md5sum`, `diff`, and `cmp`—and they matched according to all three tests.

Figure 17 is a screen shot that shows the two binaries, which are different sizes, producing output with identical MD5 hashes, when run with identical parameters. The previous figures show our modifications are minimal; the changes to `bmap.c` consisted solely of changing “map” to “m”, “carve” to “c”, “slack” to “s”, “putslack” to “p”, “wipeslack” to “w”, “checkslack” to “chk”, “slackbytes” to “sb” and “bmap” to “prog”. We changed *nothing else*, no spacing, tabbing, punctuation, or anything. The likelihood that an *entire* page of text in two programs would be formatted *identically* is extremely small.

While a “smoking gun” would have been to have the binary match exactly, we feel that this result is a strong indication that these binaries come from the same source.

© SANS Institute 2004, Author retains full rights.



```
root@localhost:tmp/mowgli/bmap-1.0.20
File Edit View Terminal Go Help
[root@localhost tmp]# ll prog
-rwxr-xr-x  1 root  root    487476 Apr 11 16:11 prog
[root@localhost tmp]# ./prog --doc man | md5sum -
cce4dba5e2718a2b8ca02098117f411f -
[root@localhost tmp]# pushd
/tmp/mowgli/bmap-1.0.20 /tmp ~/sansresults
[root@localhost bmap-1.0.20]# ll bmap
-rwxr-xr-x  1 root  root    543908 Apr 11 00:41 bmap
[root@localhost bmap-1.0.20]# ./bmap --doc man | md5sum -
cce4dba5e2718a2b8ca02098117f411f -
[root@localhost bmap-1.0.20]#
[root@localhost bmap-1.0.20]#
```

Figure 17 comparison of hash of output of two binaries

Legal implications

It is highly probable that `bmap` was run on the system in which the floppy was found. `bmap` was on the floppy that was in John Price's computer; in addition, the floppy contained a file listing (most likely illegal) MP3 download sites hidden in the slack space of another file.

But, it is *not* 100% certain. We cannot show the data on the floppy was put there by `bmap` run from *that* PC and not on (say) a home machine. A good defense attorney could introduce some doubt on tying the disk to the work machine. Having any supporting evidence, such as router flow logs, would be extremely useful to establish what was going on—for example, showing that John Price's machine established a connection to `ripped.net` during a time when John was at work.

Strictly speaking, the `bmap` program does not do anything illegal. It might violate a policy on using company machines for personal use or stating that all information is "owned" by the company and must be visible by them (no encryption, etc.). The `netcat` program implies there was a direct connection

with another machine, which might also be against company policy—for example, using machines for non-work related purposes.

The laws most likely to have been broken would involve copyright laws and computer fraud. 18 USC 2319 defines illegal distribution of copyrighted material. The computer fraud act could be violated by wiping the disk of the machine, which might be a “protected computer” under 18 USC 1030. Additional state laws may also be applicable—for example Chapter 40, Article 156 of the New York State Consolidated laws defines computer tampering (S156.20-27), unlawful duplication of computer related materials (S156.30), and criminal possession of computer related materials (S156.35). A detailed discussion of these laws, their applicability and penalties, is presented in Part III of this report.

Interview questions

A few approaches can be used in an interview to help prove the subject had installed and executed the program, depending on the nature of Mr. Price. The first and least likely to work is the straightforward approach: tell him that we can show the floppy belongs to him and that we have identified the binary, and give him a chance to talk. Questions 1 and 2 take that approach.

The second approach is to intimidate him, revealing enough information to show that he is in big trouble, and use that to get him to confess and reveal additional information, possibly making him feel betrayed by his “associate” Mike. Questions 3 and 5 take that approach.

If he appeared to be confident, a third approach would be to appeal to his inner geek, impress him with the technology and make him want to brag. Question 4 takes that approach. We could make deliberate technical errors that he might correct to show how much he knows. Question 6 takes that approach.

1. Tell me what was on that floppy. There is a directory named John and a mail message you sent to Mike. What was the program, what did it do? When did you rename bmap to prog?
2. Why did you wipe your computer? What did you put on it? You know that one format isn't enough to get rid of information. It just takes us longer and costs us more to find it, and means we're going to pursue the full penalty of law. Let me see if I can jog your memory...do these sites sound familiar: fileshares.org, convenience-city.net, emmpeethrees.com/hidden, ripped.net/down/secret.htm?
3. We've already talked to Mike. In fact, he was the one that fingered you. If you work with us, that'll help. The police might be willing to cut a deal. If not, I'm sure Mike'll be happy that you took the rap alone. You know the RIAA is looking for people to hang up for publicity. The law says, what

[ask the “bad” cop], 5 to 10 years for willful infringement of copyrighted material and 10-15 under the anti-hacking law? You’re looking at up to 25 years if you go it alone. Good friend, that Mike. Now, have you got anything you want to say about him?

4. We examined the floppy in your computer. It’s got your name on a bunch of files, so you won’t be able to claim it’s not yours. We saw netcat, found bmap, and found your list of download sites. Cute, hiding it in the HOWTO. We’ve got enough on you to prosecute. [pause] It must’ve been incredibly tedious to transfer all those files by hand, and to organize them. How could you stand to just type file names all day and move all that stuff around? Didn’t you at least throw together some scripts to make life a little easier and keep things organized?
5. Are you familiar with MAC times on a file? You know they can be used to analyze what happened on a computer? It appears that you don’t know that inode numbers are just as useful, even when someone attempts to hide their trail. Like they say, we can do this the easy way or the hard way, it’s your choice. If you help us, we’ll help you. Just tell us everything, from the beginning. And remember, we already know most of the details, so we’ll know if you try to lie.
6. Do you really understand how bmap works? Did you think no one would notice the slack space being used? If we hadn’t seen that program, we could’ve compared du and df. [At this point, he might want to correct our “misunderstanding” on how slack space works. Allow him to “teach” us and show that he understands it.]

Case information

System administrators can detect if bmap has been used on their systems to hide data in the slack space of files by running the following command sequence, as suggested in [9]:

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

This command enumerates all the files on a disk (starting with the root directory), runs the bmap command on it to check if any non-null data is stored in the file’s slack space and prints any results that match the words “has slack.” Normally, Linux stores nulls in the slack space of files. This command would be useful for system administrators to run to detect data hidden in the slack space in their disks. It would be a good indication that this binary, or a functionally similar one, were in use.

inode-based timeline analysis

We created two timelines, one based on the MAC times and the other based on the inode numbers, and compared them. The two analyses are inconsistent,

suggesting either that another machine was used (with a different time base) or that the MAC times were deliberately altered to hide the trail. We describe the timeline analyses below.

Using the timeline created by the Autopsy [1] program (included in Appendix A), we looked at the MAC times of the file named `prog`. In addition, we compared that to a list of files on the floppy, sorted by inode number (included in Appendix B). Operating systems generally allocate inode numbers in a sequence. So the sorted inode list gives us a relative timeline for the order in which files were created.

We tested how inodes are allocated on a Red Hat 9 system, creating an ext2 floppy via the `mkfs` program, then creating files, deleting some, creating more, and observing how the inode numbers were allocated. We observed the system to always allocate the lowest available inode number. Thus if files were not deleted, ordering by inodes shows the relative order in which files were created. However, out of order inodes may reflect the presence of files that were created earlier, and then deleted, freeing a (lower number) inode for reuse. For example, inode number 13 is `/Docs/DVD-Playing-HOWTO-html.tar` and inode number 15 is `Docs`. It should be noted that Windows does *not* exhibit this behavior; it does not immediately reclaim unused inodes. Also, for large ext2 disks, files in the same directory get sequential inodes and files in other directories get sequential inodes from a different group. In this case, the floppy's limited size simplifies the inode analysis.

The inode numbering inconsistency may result if a file is created first, then a directory, and then the file is moved into the directory. Alternatively, inode 13 could have been deleted after inode 15 was created, although other evidence does not support that. An inode number analysis can be used to corroborate the data in the timeline. To defeat this analysis requires creating and deleting many files throughout the time the computer is used. This technique to defeat the inode analysis does not seem to be a common practice.

The file MAC timeline analysis does *not* agree with the inode timeline analysis. In the timeline, there are many files that have an MA (modify, access) time, and then later on have a c (inode change) time. A c-time by itself implies the file was renamed, moved within the same file system, changed ownership, group, or permissions, or had its m- and a-times changed via the `touch` command. While `bmap` had to be renamed to `prog` at some point, that renaming does not explain the changes to files in `John` and many of the files in `Docs`. The m-time of the `Docs` directory is earlier than the c-times of the files in the `Docs` directory; this means the c-times on those files were *not* a result of moving the files into the `Docs` directory. Had it been due to a move, then m-time of the `Docs` directory would reflect the change as well. The c-time must have been due to a different operation performed on them.

Therefore, it seems likely that the MAC times were deliberately altered to hide the trail after the fact. The c-times will show when the m- and a-times were altered, but their previous values are lost.

Another piece of evidence is that the `bmap` tool stores the date when it was built. The Makefile creates the file `config.h` which contains various `#define` statements. Of relevance to us is the `BUILD_DATE`, which is created by the command in the Makefile:

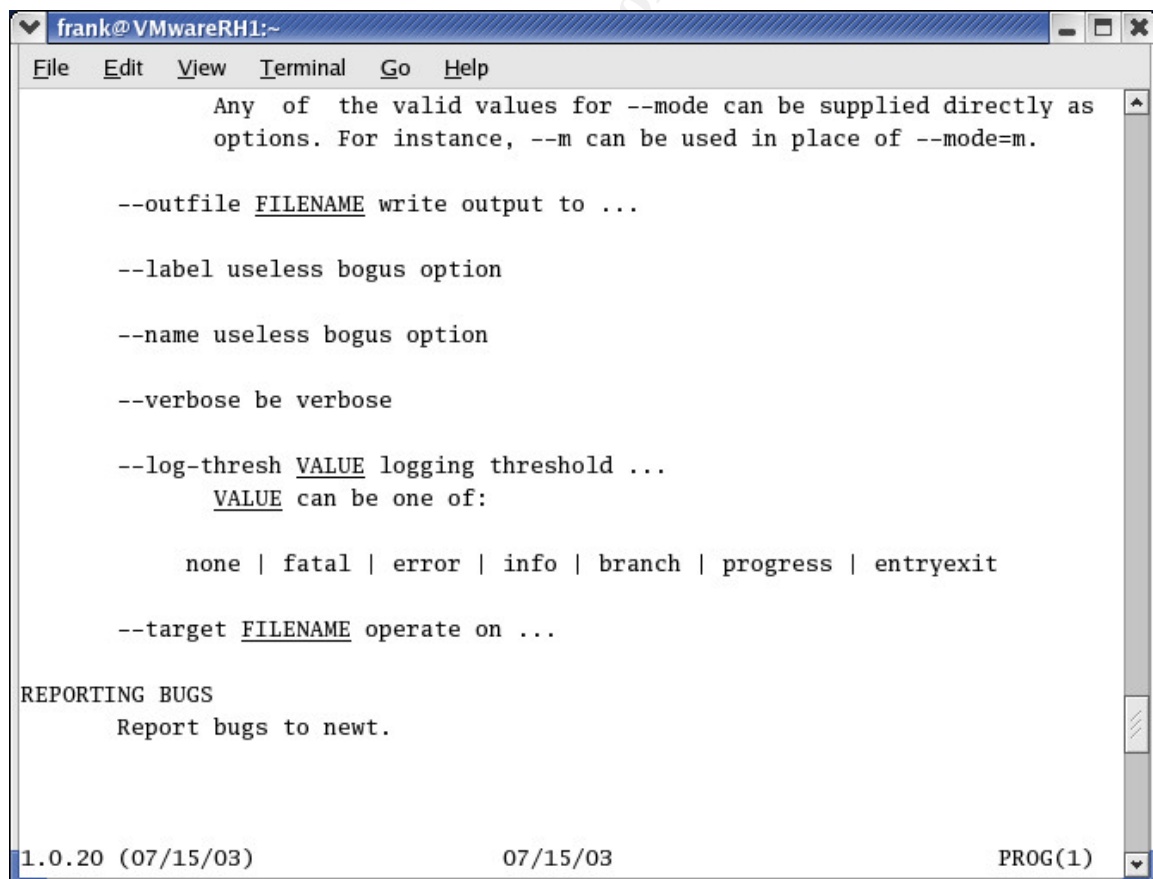
```
BUILD_DATE = $(shell date +%D)
```

followed later by

```
echo "#define $BUILD_DATE \"${BUILD_DATE}\"" >> $@
```

which sets the variable `BUILD_DATE` to the current date, of the form MM/DD/YY, in which MM is the month, DD is the day, and YY is the year. So, for example, July 15, 2003 is 07/15/03.

The `bmap` tool prints the build date when the help is displayed. Figure 18 below shows when `prog` was run with the command line arguments `--doc man`. The build date can be seen in parentheses on the bottom left side of the screen as well as at the bottom center of the screen. It is 07/15/03.



```
frank@VMwareRH1:~  
File Edit View Terminal Go Help  
Any of the valid values for --mode can be supplied directly as  
options. For instance, --m can be used in place of --mode=m.  
  
--outfile FILENAME write output to ...  
  
--label useless bogus option  
  
--name useless bogus option  
  
--verbose be verbose  
  
--log-thresh VALUE logging threshold ...  
    VALUE can be one of:  
        none | fatal | error | info | branch | progress | entryexit  
  
--target FILENAME operate on ...  
  
REPORTING BUGS  
    Report bugs to newt.  
  
1.0.20 (07/15/03)           07/15/03           PROG(1)
```

Figure 18 screen shot of help screen with prog version and date

Looking at the file timeline, we can see that the m-time of the file was July 14, 2003 at 10:24am, the c-time of the file was July 16, 2003, 2:05am, and the a-time of the file was July 16, 2:12am. If the build time in the binary is correct, then the last modification time occurs *before* the file was built. One explanation is that the m-time was changed on the 16th. However, the inode listing shows the file created immediately before `prog` was `Mikemsg.doc` and the MAC time of it is July 14 2003, 10:48. Another explanation is that the binary was built on a *different* machine with a clock set to a different time or possibly in a different time zone.

So, the timestamp on `bmap` conflicts with the build date compiled into the program, as well as the date on the message for to Mike. This suggests that either another machine was used to build `bmap` or that the file timestamps and the clock on John Price's machine were deliberately altered to cover the trail.

Several details suggest that John Price was using the organization's computing resources to distribute copyrighted material. The file `downloads` was stored in the slack space of the file `Sound-HOWTO-html.tar.gz` and contains a list of pirate sites from which MP3s can be illegally obtained (note that these are not real sites, but for the assignment, we can assume they are). Also, the message to "Mike" is from John Price. It is signed by "JP" and the properties of the file, viewable under Microsoft Word, shows the document author as John Price. In addition, the text of the message mentions that he received files "last night" (July 13, 2003) and jokes that he is "ready to rock and roll" which further supports the claim that John Price was distributing copyrighted material. The floppy contained `netcat`, which would allow him to make network connections to other machines and transfer data. Finally, the floppy contains a directory named "John," which further suggests that this is John Price's disk. It should be mentioned that many of these clues could be faked to frame John Price. However, this must be weighed against the overall consistency of the facts and how they corroborate each other.

The images are curious anomalies. The Ebay image may be a lead as to how Mr. Price distributed the material. A search on Ebay revealed a user named Johnprice, but he had not been active recently.

Additional information

The following web pages were useful during the research for this part. Link 1 provided useful information on the slack space tool `bmap`. Links 2 and 3 provide detailed information on the `ext2` file system. This was useful in understanding the physical layout of the disk. Link 4 suggested running `bmap` with the `find` command to find all files with slack space on a disk. It also has good forensic resources. Finally, Link 5 is the Legal Information Institute at Cornell University,

and provides easy to navigate, searchable, and cross-referenced web pages of the United States Code (as well as other legal sources).

1. "Linux Data Hiding and Recovery" by Anton Chuvakin, PhD.
http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html
2. "Some Notes on the Linux Kernel" by Andries Brouwer, 2003-02-01, Section 7.2, the Ext2 file system, <http://www.win.tue.nl/~aeb/linux/lk/lk-8.html#ss7.2>
3. "Design and Implementation of the Second Extended Filesystem" by Rémy Card, Theodore Ts'o, and Stephen Tweedie,
<http://e2fsprogs.sourceforge.net/ext2intro.html>
4. "Forensic Tools", <http://www.forinsect.de/forensics/forensics-tools.php>.
5. "US Code" by the Legal Information Institute,
<http://www4.law.cornell.edu/uscode/>.

© SANS Institute 2004, Author retains full rights.

Part 2 – Option 2: Forensic tool validation

Introduction

Timelines are an essential tool in forensic analyses. File MAC (modify, access, change) times are the building blocks forensic analysts use to generate them. Interpreting timelines is less precisely defined than building them, and relies on the knowledge and experience of the analysts. Anomalies in the timeline are often “footprints” of certain activities. Attackers commonly use archive programs, typically to extract files downloads onto a compromised system. The footprints left in the timeline will depend on the tools that were used, the operating system on which they were run, and even the type of file system used.

To determine what effects a particular archiving/extraction program may have had, an investigator would need to conduct an experiment on a particular configuration, with the right tools. While this is a straightforward process, it is tedious and easy to make mistakes that could contaminate the results of the tests. What is needed is an automated tool to assist an investigator to conduct these tests in a repeatable, forensically valid way.

We have created such a tool, called tar2d2.³

The following scenario describes how tar2d2 could be used. An investigator gets a disk to examine and notices an anomaly in the file system—for example, create times are more recent than modification times, modification times pre-date the system installation, and the file owner is not a local user. The investigator also notices two archive tools on the system and notes the last access time of each. He forms a hypothesis that one of the tools was used to extract files from an archive. He then uses tar2d2 to test his hypothesis on a machine running the same operating system and configuration. Another situation is that tar2d2 can be used to create a library of fingerprints of various archive programs, which can then be used to help create hypotheses to explain anomalies.

In this part, we conduct a tool validation analysis, demonstrating that the data produced by the tool is verifiable and repeatable, and showing the forensic integrity of the tool. In the process, we will describe the design, function, and use of the tool.

Scope

In order to properly test this tool, we ran it on two different operating system platforms—Windows XP Pro and Linux Red Hat 9.0. We conducted the tests

³ While technically, the name can stand for Testing of Archivers for Reference of Restored Disk Data, in reality the name is a bad pun, since the tool is a helpful little droid.

using two different archiving programs, tar and zip. Tar2d2 is a tool that helps an investigator run tests on archiving programs. It runs the tests, gathers the data in a specific order, and stores it in a consistent way. It is designed to be used in the lab, so that file system fingerprints can be observed.

The scope of testing is limited to running the tool on two different operating system platforms, and observing and analyzing the results in order to show consistent performance.

Tool description

The tool name is: tar2d2, version 1.00, written by Frank Adelstein. The tool is freely available and redistributable and can be obtained at the web site: <http://atc-nycorp.com/downloads/frank/tar2d2.pl>.

The tool is designed to help automate the process of testing archiving tools and observing their effects. Different tools have different affects—for example, an extraction program that explicitly sets the modification time on a file based on the archive may have the create time *after* the modification time. Alternatively, the granularity of the timestamps on the file system may show access times as midnight of the current date.

It can be useful for an investigator to verify the behavior of the suspected program on a computer in a particular configuration. Conducting the test by hand *can* be straightforward, but to conduct it in a forensically sound way is tedious and can be prone to contamination if the investigator is not careful. tar2d2 is designed to provide a standardized process for conducting these tests, storing the data, and comparing the data with a standard. These tests are easily reproducible and could support the testimony of an expert witness.

A more detailed description of the tool's functionality is presented later, but briefly, each experiment goes like this: the experimenter starts with a central directory, which is then archived; he then uses tar2d2 to extract files from the archive to a specified location and compare the extracted version with the original.

tar2d2 helps the forensic investigation by performing tests in a controlled, repeatable way, so investigators can determine (or verify) the effects of running archive programs. The program was designed to perform the tests in a forensically sound way and preserve useful information about the “forensics fingerprint” of the archive program under consideration. It is extensible and configurable and runs on Unix and Windows platforms.

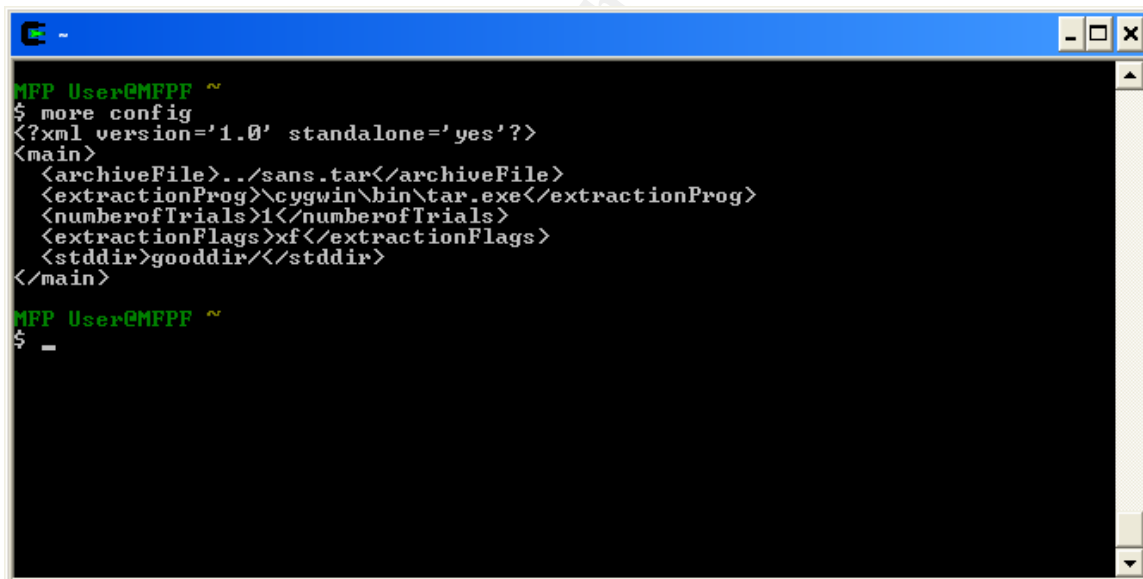
By using this tool, investigators gain a clear understanding of the forensic fingerprint of an archive tool run with specific options on a particular OS platform using a particular file system. This allows investigators to recreate the conditions

on a system under investigation and determine what effects a particular tool has. The effects include preserving or overwriting values including the MAC times, file and group IDs. In addition, the tool creates a datafile that contains all of the relevant information, including MD5 hashes of all of the files examined, in order to preserve the integrity of the results of the experiments.

Detailed description of tar2d2:

tar2d2 is invoked via the command line. The user specifies a directory to be used for datafile extraction and a filename in which the results are stored. The user can specify a configuration file, as well as “debugging” and “verbose” options.

The config file is an XML file that specifies the location of the archive/extraction program, the flags to be passed to the program, the location of the archive file, and the location of the control directory (which may, but need not be, a CD-ROM or read-only mounted file system). In addition there is a field to vary the number of tests run, but that was not used for this validation (instead, we performed multiple individual runs of the entire tool). Figure 19 shows a screen shot containing a config file that was used on a Windows XP system.



```
MFP User@MFP ~
$ more config
<?xml version='1.0' standalone='yes'?>
<main>
  <archiveFile>../sans.tar</archiveFile>
  <extractionProg>\\cygwin\\bin\\tar.exe</extractionProg>
  <numberOfTrials>1</numberOfTrials>
  <extractionFlags>xf</extractionFlags>
  <stdDir>gooddir</stdDir>
</main>
MFP User@MFP ~
$
```

Figure 19 screen shot of config file on Windows XP Pro

Figure 20 shows the configuration file used for Linux.

```
<?xml version='1.0' standalone='yes'?>
<main>
  <archiveFile>/tmp/sans.star</archiveFile>
  <extractionFlags>xf</extractionFlags>
  <extractionProg>/usr/bin/star</extractionProg>
  <numberOfTrials>1</numberOfTrials>
  <stdDir>/mnt/hack/forensic_challenge_mount</stdDir>
</main>
```

Figure 20 tar2d2 config file for Linux

If no configuration file is specified, a default is used and the file `config` is created. While this is unlikely to guess the location of the files, it is a fast way to create a conforming XML file which can be edited to change the path specifications as required.

The tool does *not* verify that the control directory is mounted read-only. The investigator must ensure that during the setup, otherwise the access times on the files and directories in the control directory will be changed.

The tool then gathers information about the control directory, iterating recursively through all the files in it and writes the data to an output file (specified by the user with “.std” appended to the end of the name). The file persists but is only useful for debugging or comparison purposes (refer to the conclusions for suggestions on improvements). A detailed description of the data gathered is presented later.

The tool then verifies that the command line specified directory to use for the extraction of the files is empty. The program terminates with an error message if the directory is not empty. Then it changes to that directory and runs the specified archive program. Output and errors go to STDOUT and STDERR, which can be captured by simple redirection.

After extraction, the tool gathers data on the extracted files in the same manner that it collected information on the control data. The tool gets a list of all of the files in the directory and performs a `stat()` library call on each one in turn, and saves the data. The data consists of the name, m-time, a-time, and c-time, file size, inode number, mode (permissions), number of links, owner ID, group ID, rdev (major and minor numbers if it is a device file), block size, and number of blocks. In addition, the program computes the MD5 hash of the file (the MD5 field for directories are listed as “0”).

If the file is a directory, then *after* obtaining the information, the tool recursively descends into that directory and repeats the process.

After gathering the data on the extracted files, the tool compares the data from the extracted files in the test directory to the files in the control directory. The tool compares each attribute (i.e., m-time, size, etc.) in each file in the test directory to that of the control group. If the values are the same, the value is considered *preserved*, and if the values differ, it is considered *overwritten*. The tool keeps a count of how many times each attribute is preserved or overwritten. The tool then writes summary information, which includes two categories: overwritten and preserved. Each category contains a list of the attributes and a count of how many times each was overwritten or preserved. In addition, the attribute *numberoffiles* is a count of the total number of files and directories processed. As a check, the count for each attribute in preserved and overwritten should sum to numberoffiles.

By considering the preserved and overwritten MAC time attributes, an investigator can quickly verify what an archive program is doing when it performs an extraction. Other attributes such as owner and group may also be useful. The file size and MD5 hash can be used to support the forensic integrity of the experiment by verifying that the

Certain attributes will never be preserved, such as the inode number in Linux. Others will always be preserved, such as the name and MD5 hash. The behavior of some attributes will differ between Linux and Windows, due to library implementations. For example, blocksize and blocks will be empty under Windows.

Finally, the tool writes out the data file in XML. Examples of the data files are included in Appendix C and Appendix D.

Since this is a perl script, it uses the dynamic libraries that perl uses. Figure 21 shows a list of the dynamic libraries used by perl, version 5.8.0, on our Red Hat 9 Linux system.

```
# ldd /usr/bin/perl
libperl.so => /usr/lib/perl5/5.8.0/i386-linux-thread-
multi/CORE/libperl.so (0x40017000)
libnsl.so.1 => /lib/libnsl.so.1 (0x4015c000)
libdl.so.2 => /lib/libdl.so.2 (0x40171000)
libm.so.6 => /lib/tls/libm.so.6 (0x40174000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40196000)
libc.so.6 => /lib/tls/libc.so.6 (0x42000000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x401a4000)
libutil.so.1 => /lib/libutil.so.1 (0x401d1000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Figure 21 dynamic libraries used by perl under Linux

In addition, tar2d2 uses the following perl packages: `Getopt::Long`, `Digest::MD5`, `XML::Simple`, `Cwd`, `File::Spec::Functions`, and `File::Glob`. `XML::Simple` is the only package not provided with the standard perl installation and can be installed with the command shown in Figure 22.

```
# perl -MCPAN -e shell
cpan> install XML::Simple
```

Figure 22 command to install perl XML package

While the tool *can* be run from a CD-ROM, it is not necessary, since it is intended to be run on the investigator's machine, which should be in a known (and controlled) state. Similarly, there is no need to statically link perl (which can be difficult). To ensure tar2d2 is used in an evidentiary sound way, the investigator must be able to document the state of his machine, including what was installed on it and *not* use the machine for arbitrary programs (such as web browsing and reading mail). A simple approach is to use a virtual machine via VMware. VMware has the capability of taking a snapshot of the state of a virtual machine and later to restore the machine to that state. The state includes the disk, memory, and screen. If the investigator maintains an MD5 hash of the files associated with the snapshot, he can ensure the snapshots have not been modified. By performing and documenting this process, the investigator can ensure that tar2d2 is used in an evidentiary sound way.

Test apparatus

In this section, we describe the testing environment, including the OSs used (version and configurations), the format of disk (e.g., NTFS, ext2), and tools used to conduct the tests (e.g., stat, cygwin).

Red Hat Linux test system

We used GNU tar version 1.13.25, which was the same version that was used on the windows platform.

```
#uname -a
Linux localhost 2.4.20-8 #1 Thu mar 13 17:54:28 EST 2003
i686 i686 i386 GNU/Linux
# cat /etc/issue
Red Hat Linux release 9 (Shrike)
```

The machine used is a server installation with all packages selected and installed. There are 1396 RPM packages listed when the command `rpm -qa` via the command. Due to the large number of entries, this list is not included in this report. The file system is an ext2 format.

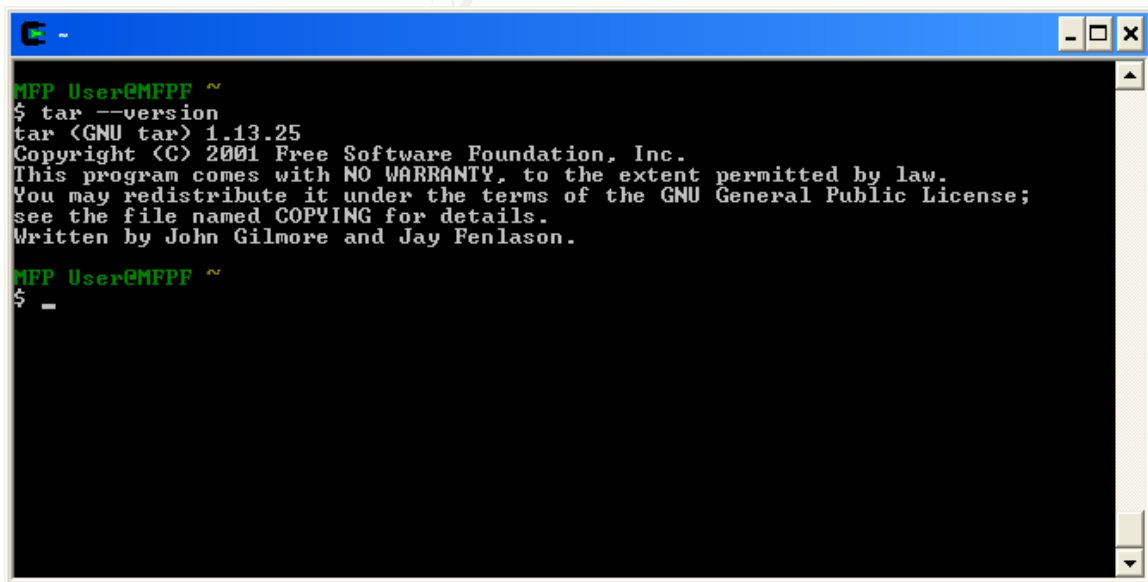
The extraction program used to test tar2d2 on Linux was GNU tar version 1.13.35 and is included in the standard Red Hat 9 distribution.

Windows XP Pro test configuration

The Windows XP Pro system is configured as follows:

Kernel: Microsoft Windows XP, Uniprocessor Free
Service pack: 1
System root: C:\WINDOWS
Build number: 2600
Registered organization:
Processor #1: 1395 MHz
Physical memory: 535740416 bytes (510.92 MB)
Drive information:
C:\ Fixed IBM_PRELOAD NTFS 24783728640 (23635MB)
24783728640 (23635MB) 36485373952 (34795MB)
D:\ CD-ROM - - UNKNOWN (UNKNOWN MB) UNKNOWN
(UNKNOWN MB) UNKNOWN (UNKNOWN MB)
Hotfixes:
A total of 46 hotfixes were installed.

The tool was tested using the C: drive, which is an NTFS disk. The extraction program used to test tar2d2 was GNU tar version 1.13.25, as supplied by the standard Cygwin installation. Figure 23 shows a screen snapshot with tar invoked with `--version`.



```
MFP User@MFPF ~
$ tar --version
tar (GNU tar) 1.13.25
Copyright (C) 2001 Free Software Foundation, Inc.
This program comes with NO WARRANTY, to the extent permitted by law.
You may redistribute it under the terms of the GNU General Public License;
see the file named COPYING for details.
Written by John Gilmore and Jay Fenlason.
MFP User@MFPF ~
$ _
```

Figure 23 Screen shot showing the version of tar used on Windows XP

Environmental conditions

The tests were conducted on two machines, one running Red Hat 9.0 Linux and the other running Windows XP Pro. The machines are connected to a local network, assigned an IP address via DHCP, and the connection to the Internet goes through a router/firewall that provides Network Address Translation (NAT). The IP addresses are in the 192.168.1.* range. Both systems were connected to the network via an IEEE 802.11b wireless Ethernet card. The Linux system used Lucent Technologies Orinoco Gold PCMCIA card, and the Windows system used an Intel(R) PRO/Wireless LAN 2100 3B Mini PCI Adapter built-in card. The network connectivity of the machines should not be significant and the disks used were local and had no other users. The tests were conducted either on the weekends or sufficiently late at night that no one else was on the network.

The Red Hat machine runs on an IBM R32 Thinkpad running a 1.8GHz Pentium 4 processor with 40G of disk and 768M of memory. The Windows XP Pro machine is an IBM T41 Thinkpad running a 1.4GHz Pentium M processor with 30G of disk and 512M of memory.

Description of the procedures

The testing is performed using a single PC. Tests were run on two platforms, a laptop running Linux and a laptop running XP.

Preparation of the platforms includes installing Perl and the XML::Simple library package. In addition, we created the following set of test files with properties related to their names:

- `readfile`
- `writefile`
- `chmodfile`
- `movedfile`
- `firstdirectory`
 - `readsubdirefile`
- `secondsubdirectory`
 - `movedsubdirfile`

The file `readfile` was created and then at a later point in time it was read, so its access time would be after its modification time. Similarly, `writefile` was created and *not* read. `chmodfile` was created and then had the permissions changed via the `chmod` command. `movedfile` was renamed via the `mv` (move command).

Two directories were created, `firstdirectory` and `secondsubdirectory`. A file `readsubdirfile` was created in that directory and then subsequently read after it was written, similar to `readfile` in its parent directory. And finally,

`movedsubdirfile` was created in the parent directory and moved into `secondsubdirectory`.

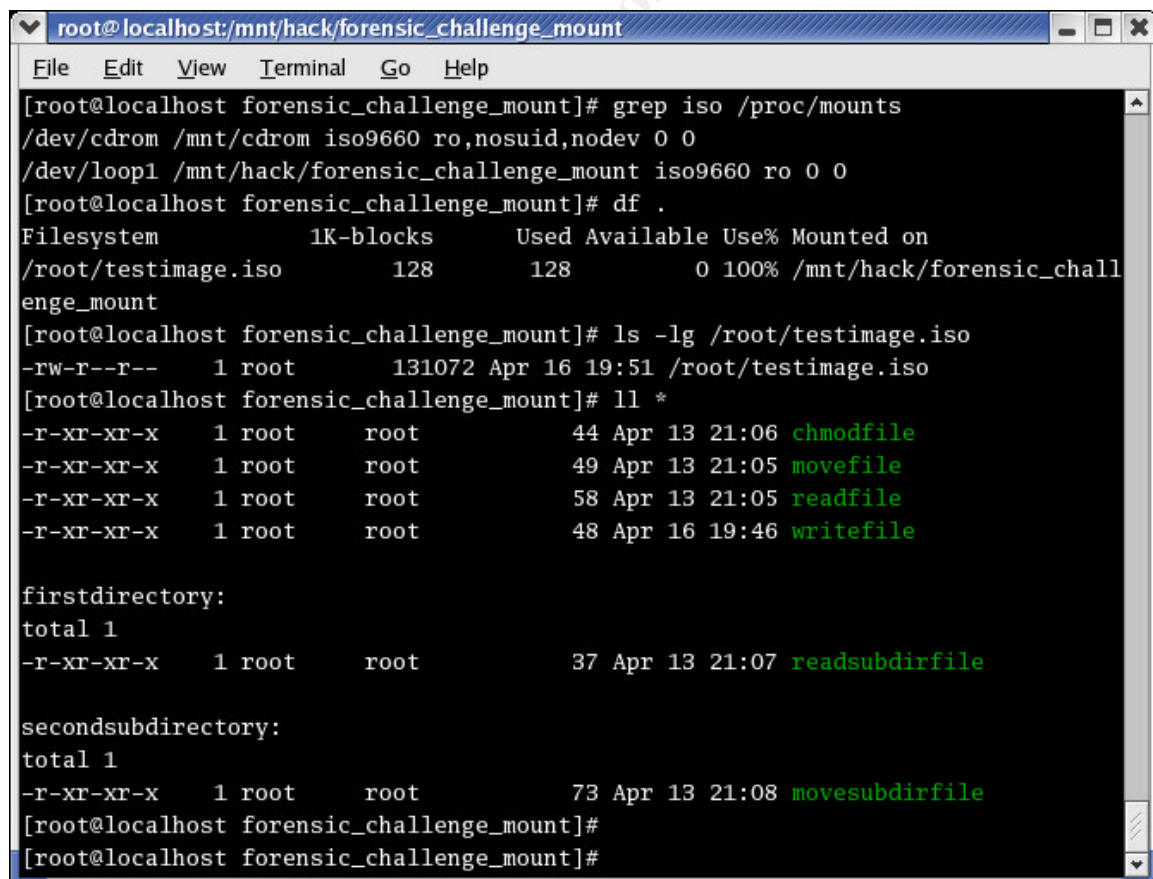
This represents a selection of common operations performed on files and would be reflected in their timestamps. The file system was then turned into an ISO 9660 image with Joliet extensions by using the command:

```
mkisofs -J -R -o testimage.iso dirname
```

That command created an ISO image of the directory subtree starting at “dirname” which contained the previously mentioned files. We then tested the integrity of the ISO file with the command:

```
mount -o loop,ro testimage.iso mount_point
```

Figure 24 shows a screen shot of the setup. The first command shows the ISO is mounted read-only via the `/dev/loop1` device on the mount-point `/mnt/hack/forensic_challenge_mount`. The second command, `df .`, shows the current directory is a mount of the file `/root/testimage.iso`, which is the file we created with `mkisofs`. Finally, the files under the directory are shown.

A terminal window titled 'root@localhost:/mnt/hack/forensic_challenge_mount' with a menu bar (File, Edit, View, Terminal, Go, Help). The terminal shows the following commands and output:

```
[root@localhost forensic_challenge_mount]# grep iso /proc/mounts
/dev/cdrom /mnt/cdrom iso9660 ro,nosuid,nodev 0 0
/dev/loop1 /mnt/hack/forensic_challenge_mount iso9660 ro 0 0
[root@localhost forensic_challenge_mount]# df .
Filesystem            1K-blocks    Used Available Use% Mounted on
/root/testimage.iso      128         128          0 100% /mnt/hack/forensic_challenge_mount
[root@localhost forensic_challenge_mount]# ls -lg /root/testimage.iso
-rw-r--r--    1 root      131072 Apr 16 19:51 /root/testimage.iso
[root@localhost forensic_challenge_mount]# ll *
-r-xr-xr-x    1 root      root           44 Apr 13 21:06 chmodfile
-r-xr-xr-x    1 root      root           49 Apr 13 21:05 movefile
-r-xr-xr-x    1 root      root           58 Apr 13 21:05 readfile
-r-xr-xr-x    1 root      root           48 Apr 16 19:46 writefile

firstdirectory:
total 1
-r-xr-xr-x    1 root      root           37 Apr 13 21:07 readsubdirfile

secondsubdirectory:
total 1
-r-xr-xr-x    1 root      root           73 Apr 13 21:08 movesubdirfile
[root@localhost forensic_challenge_mount]#
[root@localhost forensic_challenge_mount]#
```

Figure 24 read-only ISO image of test files

It should be noted that since this is mounted read-only, nothing will change on the mounted file system, including any of the time stamps on the files.

We then created a tar file via the command `tar cf /tmp/sans.tar .`. Then the config file was created and the empty directory to hold the extracted files, named `blarg`, was created. The code was run via the command:
`tar2d2 -c config blarg data.out`

The data file was then analyzed based on the approval criteria described in the next section. The contents of the directory were removed and the process was repeated, saving the output to a different file. The program was run three times.

Finally, we verified that `testimage.iso` had not changed (been modified) since it was created, by looking at its last modification time and its MD5 hash (`192d135a51f53538eba068d45ce38bab`).

The entire process was repeated on a Windows XP system, using a CD-ROM created from the `testimage.iso` file.

The results of the output are the “data.out” files. The files are given meaningful names, such as `linux.out` and `xp.out`. When multiple runs are done to show the files did not change, a number is added to the name, such as `linux1.out`, `linux2.out`, etc.

Criteria for approval

There are several criteria for approval for `tar2d2`.

1. The data it generates must be correct. The timestamps, MD5 hashes, etc. all must be correct.
2. `tar2d2` must examine and include all files extracted by the archive program.
3. The results must be consistent and repeatable. Subsequent runs must produce consistent results. Note that *identical* results are *not* to be expected, since inode numbers and timestamps may change from run to run; but the fundamental data it reports, and the reports that the data *has* changed in comparison to the control data set must be the same from run to run.

The evaluation addresses these criteria by the experiments. In addition, the correctness was further supported by the step-by-step description of the source code presented in the Tool Description section.

In addition, we performed a similar procedure to analyze the `tar` and `star` programs by hand. This allowed us to subjectively compare performing the

process by automation versus by hand, in terms of ease of use, accuracy, and repeatability.

Data and results

First, the code was written to perform its tasks in a repeatable and forensically valid way. For example, data is preserved by recording all of the file attributes, including MAC times, *before* anything is done to the files. The MD5 hash is computed before obtaining the MAC times, so the tool records the original access time of the file.⁴ In addition, the tool records the MAC times for a directory *before* recursively processing that directory; so again, the tool records the original access time of the directory.

Correctness test: We ran “stat” by hand on the files analyzed by tar2d2 and then used the 1-line perl script shown in Figure 25 to translate the timestamps into a human-readable format:

```
perl -e 'print localtime($ARGV[0]) . "\n" ' timestamp
```

Figure 25 perl 1-liner to translate epoch time into a human-readable date

Note that *timestamp* is replaced by the numeric value in the *atime*, *ctime*, or *mtime* tag in the result file. We ran md5sum on the files to confirm that the values in the *MD5* field is correct.

Ensure all files were processed: We verified that the file data.out contained a *file* tag for all 8 entries in the tar file which is equivalent to the ISO file. There are two directories, *firstdirectory* and *secondsubdirectory*, four files at the top level: *chmodfile*, *movefile*, *readfile*, and *writefile*, and one file under each of the subdirectories, *firstdirectory/readsubdirfile* and *secondsubdirectory/movesubdirfile*.

By executing the command “`grep name data.out`” we could easily determine how many files were in the output (by searching for “name” tags). In addition, because there were only 8 entries, we examined the entire file by hand as well.

To test the repeatability, we ran the program 3 times in a row, saving the resultant data files and then comparing them. This was done under Linux and Windows to verify that the results were consistent.

⁴ Recall that this tool is designed to be used in a laboratory to analyze archiving programs; it is *not* intended to operate on the forensic data of a compromised machine itself. So changing the a-time of files the tool creates is not significant, provided that it records the original a-times, as well as other data.

Appendix C and Appendix D includes the XML output file from running tar2d2 on the sample files under Linux and XP, respectively.

Analysis

tar2d2 provides two types of data to the investigator. The first type is the raw data on the files and directories and the second is the summary information. The raw data is useful for verifying the functioning of the tool—for example, showing the MAC times are correctly recorded. The raw data can also be used to gain insight into some of the low level OS functioning. The order in which inodes are allocated within a single directory and across directories is shown by examining the raw data.

The summary data is more useful to the investigator. By examining the “preserved” and “overwritten” attributes, an investigator can quickly determine what file attributes have changed during extraction. In addition, the output file is in XML format, which can be parsed easily by other scripts, either for analysis or entry into a database.

The preserved and overwritten attributes help characterize the fingerprint of an extraction tool. While some tools may not leave distinguishing fingerprints on the file system, others may. This data can then be used to support an investigator’s hypothesis that a particular tool was used to explain artifacts left in a file system.

The analyses conducted for this practical are based on the acceptance criteria described above. The raw data consists of MAC times, file hashes, directory listings, file sizes, as well as experiment configuration data (OS platform, program, filesystem, etc.).

Presentation

The data saved by tar2d2 are stored in XML, which is a general format that is easily parsed and read in by other tools. The perl library XML::Simple allows an XML file to be read in directly into a data structure consisting of direct values of strings and numbers (floats and integers), arrays, and associative arrays (hashes). In addition, to the raw data stored in the `<file>` `</file>` tags, the `<preserved>` `</preserved>` and `<overwritten>` `</overwritten>` tags contain a set of tags indicating which attributes in the extracted file were identical to the control file (the read-only ISO image) and which were modified. Each subtag under preserved and overwritten contains a count of how many files preserved or overwrote that attribute. Figure 26 shows an example in which the *ctime* and *atime* attributes were the only attributes changed. The attribute *numberoffiles* represents the number of files processed, so it serves as a check that all files were processed. In general, all files should fall into one group or

another, but it is not required. For example, directories may be treated differently than regular files.

```
<numberoffiles>8</numberoffiles>
<preserved>
  <dev>8</dev>
  <mode>8</mode>
  <nlink>8</nlink>
  <MD5>8</MD5>
  <ino>8</ino>
  <size>8</size>
  <name>8</name>
  <rdev>8</rdev>
  <mtime>8</mtime>
  <gid>8</gid>
  <uid>8</uid>
  <blksize>8</blksize>
  <blocks>8</blocks>
</preserved>
<overwritten>
  <ctime>8</ctime>
  <atime>8</atime>
</overwritten>
```

Figure 26 sample XML summary tags

The summary information mentioned above could be used to validate a hypothesis about the traces left by running an archive program. An investigator would create a hypothesis to explain particular artifacts (say, time stamp anomalies) after observing that an archive program had been run recently (based on its access time). To help support the hypothesis, he could run tar2d2 to show the archive program leaves the same traces that were observed. Similarly, it could be used to refute a claim that such a program was used by showing the archive program does *not* leave such traces.

Explaining the output of the tool to a court would be challenging. Anything involving the technical details of a file system, the time stamps, creation or (worse) inode modification change time can quickly lead to glazed over expressions and a lack of understanding, or worse, a misunderstanding (perhaps even a sentiment that the prosecution is trying to “hide” the truth or misdirect the court by presenting all these “facts” and numbers).

The less time spent on an explanation, the better. The investigator would need to mention that he looked at files on a computer and that the operating system records whenever a user or program reads or writes files (leave off c-time unless

needed). These are the timestamps. The investigator has reconstructed the actions of the defendant based on evidence *in* the files and the timestamps *on* the files. The claim is that a certain program was run. The investigator can say that he has *tested* the validity of the claim in the lab, using a scientifically and forensically valid method, and it shows what evidence would be left by running the tool. This information then supports or refutes the claims.

Conclusions

We feel the testing successfully demonstrated the validity and repeatability of the tool. In addition, performing the task by hand emphasized the need for an automated, easily repeatable process. It is easy to make mistakes, perform steps out of order, or accidentally contaminate the data with a stray `ls` command. In addition, recording the data in a consistent, machine-readable format is tedious and error prone. Automating the process is highly desirable.

A question remains as to the overall usefulness of such a tool. A colleague who is the security administrator at a large university said, in private communication, that a list of fingerprints of archive tools would be useful. However, the usefulness of a tool will be decided by those in the field “in the trenches.”

The tool is designed to produce forensically sound data. It can *not* be used for incident response on original data. The tool is intended to reproduce the effects of archive tools to help explain the anomalies that might be observed in a MAC timeline. It must be used in a controlled laboratory setting using calibrated test data that is either provided with the tool or created by the investigator.

Limitations: tar2d2 relies on some of the abstractions defined in the perl libraries to increase its portability. Because of this, some data is not obtained on Windows machines. The most important data—specifically file MAC times, sizes, and MD5 hashes—are correctly obtained, but other data including permissions, owner and group, and inode number are not. The investigator must be aware what data is valid under Windows when determining a fingerprint.

A caveat: the author of this report is the author of the tool. A study made by an independent third party would be warranted to validate the results presented here.

Potential improvements: tar2d2 focuses on determining the fingerprints of extraction tools, since such tools are often used to exploit compromised machines once software has been downloaded, such as root kits and IRC servers. Files are commonly gzipped tar files, tar files, or zip files, so this is useful. However, with a little additional work, tar2d2 could support analyzing archive *creation* as well as extraction. For example, depending on the tool and the flags used, an extraction program may preserve the a-times at the expense of c-times. Noting the presence of this artifact on all files associated with a

popular archive might suggest that an archive was *built* on a machine under investigation. Being able to verify that the tools on a given machine behave in the presumed way would be useful.

The `.std` file is only useful as a debugging tool. The program should be modified so that it is created only when the debugging flag is enabled.

Windows uses a more complex mechanism for file permissions: access control lists (ACLs). A useful addition would be supporting Windows ACLs, although that requires non-portable OS-specific code.

Additional Information

We discovered additional information during the creation and testing of this tool.

Interesting details (re)discovered during the testing:

- Linux allocates inodes in a regular, repeatable way. Windows allocates them in a (seemingly) random way.
- `star -k` preserves all 3 MAC times. An impressive trick (performed by temporarily altering the system time).

We focused on data file *extraction*, since investigators typically will obtain disks in which archives were unpacked (root kits, etc.). Archive programs also leave fingerprints when they create an archive. There are flags that determine whether they will affect the a-time or not (by changing the c-time). Further experiments on archive creation would be interesting, and tar2d2 would support those experiments with minor modifications.

Windows handles ISO file systems poorly, often ignoring MAC times on the media. An improvement to tar2d2 would be to have it create the files with the desired properties before the tests run.

Part 3 – Legal issues of incident handling

Question A

This question asks what laws were broken, if John Price was distributing copyrighted MP3s.

Federal Law

By distributing copyrighted material, John Price is committing a criminal infringement of a copyright. Title 17, Chapter 5 of the US Code defines “Copyright Infringement and remedies” and specifically, Section 506(a) defines “Criminal offenses” as follows [14]:

- (a) Criminal Infringement -
 - Any person who infringes a copyright willfully either –
 - (1) for purposes of commercial advantage or private financial gain, or
 - (2) by the reproduction or distribution, including by electronic means, during any 180-day period, of 1 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of more than \$1,000,
- shall be punished as provided under section 2319 of title 18, United States Code. For purposes of this subsection, evidence of reproduction or distribution of a copyrighted work, by itself, shall not be sufficient to establish willful infringement.

Title 18, Part 1, Chapter 113 of the US Code covers “Stolen Property.” Section 2319 [15] defines “Criminal infringement of a copyright” as follows:

- (a) Whoever violates section 506(a) (relating to criminal offenses) of title 17 shall be punished as provided in subsections (b) and (c) of this section and such penalties shall be in addition to any other provisions of title 17 or any other law.
- (b) Any person who commits an offense under section 506(a)(1) of title 17 –
 - (1) shall be imprisoned not more than 5 years, or fined in the amount set forth in this title, or both, if the offense consists of the reproduction or distribution, including by electronic means, during a

180-day period, of at least 10 copies or phonorecords, of 1 or more copyrighted works, which have a total retail value of more than \$2,500;

(2)

shall be imprisoned not more than 10 years, or fined in the amount set forth in this title, or both, if the offense is a second or subsequent offense under paragraph (1); and

(3)

shall be imprisoned not more than 1 year, or fined in the amount set forth in this title, or both, in any other case.

To prosecute under 18 USC 2319, it must be shown that the more than 10 copies of the copyrighted material were distributed and that that represents more than \$2,500 of value. At roughly \$20 per CD, Mr. Price would need to have distributed at least 125 copies. A first offense is punishable by up to 5 years in prison, and subsequent offenses up to 10 years. Legal counsel should be sought to determine what is necessary to establish “willful infringement” since “evidence of reproduction or distribution of a copyrighted work” is not sufficient to establish this.

If it was not “willful infringement,” 18 USC 2319(c) defines a punishment of 3 years imprisonment, fines, or both, for the first offense, if 10 or more copies having a total retail value of \$2,500 or more were distributed. Subsequent offenses are punishable by up to 6 years imprisonment. The minimum offense, requiring a total retail value of more than \$1,000 and at least one copy distributed, is punishable by no more than 1 year imprisonment, fines, or both. Again, assuming a \$20 retail price per CD, this means Mr. Price is culpable under the law if 50 copies were distributed.

While “Criminal infringement of a copyright” is the most obvious charge, it is not the only one. The Computer Fraud and Abuse Act (18 USC 1030) [16] may also be applicable. It was stated that Mr. Price was distributing the material on a “publicly available system.” Mr. Price’s PC was on the Internet. Therefore, if he was using his PC to distribute copyrighted material, it may be considered a “protected computer” under the definition in 18 USC 1030(e)(2)(B), which includes a computer “which is used in interstate or foreign commerce or communication, including a computer located outside the United States that is used in a manner that affects interstate or foreign commerce or communication of the United States.” A publicly available computer on the Internet should fall under the definition of protected computer.

18 USC 1030(a)(5)(A)(i) states that this section applies to whoever “knowingly causes the transmission of a program, information, code, or command, and as a result of such conduct, intentionally causes damage without authorization, to a

protected computer.” It was stated that “Mr. Price was able to wipe the hard disk of his office PC before investigators could be deployed.” If Mr. Price did not merely delete files from his machine but performed a low level format of his disk, wiping out everything contained on the disk, he was knowingly causing intentional damage to his computer. He can be prosecuted under section 1030, if the total damage is \$5,000 or more, as defined in 18 USC 1030(a)(5)(B)(i). This includes the cost of the time spent restoring his machine to service, in addition to the value of the data that was destroyed.

State Law (New York State)

In addition, several different New York⁵ state laws are relevant – specifically, computer crimes are defined under Article 156, Title JA156, of the Penal code of the New York State Consolidated Laws [12]. The specific sections that are relevant are S156.20, S156.25, and S156.26 “computer tampering,” S156.30, “unlawful duplication of computer related material,” and S156.35, “criminal possession of computer related materials.”

Section 156.20, 156.25, and 156.26 define “computer tampering” in the fourth, third, and second degrees, respectively. 156.20 states:

S 156.20 computer tampering in the fourth degree.

A person is guilty of computer tampering in the fourth degree when he uses or causes to be used a computer or computer service and having no right to do so he intentionally alters in any manner or destroys computer data or a computer program of another person.

Computer tampering in the fourth degree is a class A misdemeanor.

This is, essentially, a “catch-all” that can be used for any intentional altering or destruction of data or programs on a computer. Clearly, John Price destroyed data by wiping his desktop hard disk. A more serious crime is computer tampering in the third degree, which is defined as follows:

S 156.26 Computer tampering in the third degree.

A person is guilty of computer tampering in the third degree when he commits the crime of computer tampering in the fourth degree and:

1. he does so with an intent to commit or attempt to commit or further the commission of any felony; or
2. he has been previously convicted of any crime under this article or subdivision eleven of section 165.15 of this chapter; or
3. he intentionally alters in any manner or destroys computer material; or
4. he intentionally alters in any manner or destroys computer data or a computer program so as to cause damages in an aggregate amount exceeding one thousand dollars.

Computer tampering in the third degree is a class E felony.

⁵ Since the author lives in New York State, these are the only state laws that are considered for this practical.

If the damage caused by John Price by wiping the disk was more than \$1000, he has committed computer tampering in the third degree. This would include the cost of the programs and the time required to repair the damage (and restore the programs).

In addition, if the data was in some way sensitive, it might be considered "computer material," which S156.00 (5) defines as a computer program or data which (a) contains medical records, (b) contains records maintained by the state, or (c) "is not and is not intended to be available to anyone other than the person or persons rightfully in possession thereof or selected persons having access thereto with his or their consent and which accords or may accord such rightful possessors an advantage over competitors or other persons who do not have knowledge or the benefit thereof." There is no minimum cost of damage required for computer tampering in the third degree. Finally, and more seriously, is computer tampering in the second degree, defined as follows:

S 156.26 Computer tampering in the second degree.

A person is guilty of computer tampering in the second degree when he commits the crime of computer tampering in the fourth degree and he intentionally alters in any manner or destroys computer data or a computer program so as to cause damages in an aggregate amount exceeding three thousand dollars.

Computer tampering in the second degree is a class D felony.

It seems unlikely that the damages would exceed \$3000, but that is not out of the question. If the investigation was conducted internally, then it is possible that the costs of the investigation may contribute towards the \$3000 minimum. Again, legal counsel would need to be sought.

If it can be shown that John Price downloaded MP3s illegally, Section 156.30 and 156.35 may apply. They state:

S 156.30 Unlawful duplication of computer related material.

A person is guilty of unlawful duplication of computer related material when having no right to do so, he copies, reproduces or duplicates in any manner:

1. any computer data or computer program and thereby intentionally and wrongfully deprives or appropriates from an owner thereof an economic value or benefit in excess of two thousand five hundred dollars; or
2. any computer data or computer program with an intent to commit or attempt to commit or further the commission of any felony.

Unlawful duplication of computer related material is a class E felony.

S 156.35 Criminal possession of computer related material.

A person is guilty of criminal possession of computer related material when having no right to do so, he knowingly possesses, in any form, any copy,

reproduction or duplicate of any computer data or computer program which was copied, reproduced or duplicated in violation of section 156.30 of this article, with intent to benefit himself or a person other than an owner thereof. Criminal possession of computer related material is a class E felony.

The illegal MP3s would seem to constitute data John Price knowingly possessed with the intent to benefit himself rather than the owners. And by downloading them, he created an illegal duplicate of the computer data (MP3s). Most likely, the MP3s would need to be recovered to show 156.30 and 156.35. It is likely that enough evidence exists to obtain a search warrant of John Price's home. If he has a computer at home, it may contain some of the missing pieces. 156.30 and 156.35 are self-referential and not very clear – it is essential that legal counsel provide guidance on the interpretation of the laws.

Question B

This question asks what steps should be taken upon discovering the information on a system.

As an employee, Mr. Price should have had to sign an employment agreement allowing his employers access to his data. Most likely, the company would not be considered a public provider; but even so, a signed statement would permit searching for Mr. Price's data, as provided by the Electronic Communication Privacy Act (ECPA) 18 USC 2701(c)(3) and 2703(c)(3)(C). Therefore, a representative of the company should be able to investigate Mr. Price's data – for example, his email.

In addition, the company should have an incident handling policy that provides guidance on the next steps to take, who to contact (corporate legal, internal law enforcement, local law enforcement, etc.).

Corporate legal counsel should be consulted about interpreting the laws. But in general, evidence should be gathered (via the exception to the ECPA of prior permission, provided by 18 USC 2701(c)(3) and 18 USC 2703 (c)(3)(C) before contacting the appropriate law enforcement office. Once enough data has been collected, then an appropriate law enforcement agency should be contacted. Note that we assume the copyrighted material are MP3s (as opposed to, say, child pornography, which is discussed in Question D).

If a relationship to a particular (local) law enforcement office already exists, that would most likely be a reasonable starting point for first contact, even if it is to seek a recommendation for another office. For copyright piracy, the department of Justice's web site recommends contacting the FBI local office or, if the material was imported, the US Customs Service [18].

Question C

This question asks what steps should be taken to ensure the evidence collected will be admissible if legal action is pursued in the future.

First, the evidence should be preserved via a cryptographic hash, such as MD5 or SHA1. A hash should be taken of all of the files individually, as well as collectively (i.e., creating a tar or zip file of all of the individual evidence files and taking a hash of the zip or tar file). The hash should be printed, signed and dated. Then, all the evidence should be kept in a secure location, such as a safe or locked cabinet. This helps preserve the chain of custody. A record must be kept of anyone who has had access to the data. The best approach is to limit the number of people who had access to the data to a minimum, preferably one.

The Daubert Test, established in 1993,⁶ is a set of criteria to determine if evidence gathered by a particular technique is admissible. [20] summarizes the Daubert criteria as:

1. whether the theory or technique can be and has been tested
2. whether it has been subjected to peer review and publication
3. the known or potential error
4. the general acceptance of the theory in the scientific community
5. whether the proffered testimony is based upon the expert's special skill

Essentially, this means that forensic evidence obtained in the investigation must be based on current “best practices” in digital forensics, such as [4], or based on the investigator's skill. If the latter is the case—e.g., the investigator has created a new method for extracting evidence—care must be taken to document the process, as well as potential sources of errors or error rates (e.g., bit error rates in copying data using a wireless card).

Question D

This question asks the differences in actions required if it was discovered that John Price was distributing child pornography.

Title 18, Part 1, Chapter 110, sections 2251 - 2260 of the United States Code covers “Sexual exploitation and other abuses of children” and specifically, 18 USC 2252A[17] defines “Certain activities relating to materials constituting or containing child pornography.” If John Price was distributing child pornography, 18 US 2252A would apply. It defines 5 types of activities in 2252A(a)(1) – (a)(5), which we summarize below.

The first four categories of activities relating to child pornography are defined as: (a)(1) transporting; (a)(2) receiving or distributing; (a)(3) reproducing; and (a)(4) selling or possessing with intent to sell. Section 2252A(b)(1) defines the penalty

⁶ Daubert v. Merrell Dow Pharmaceuticals (92-102), 509 U.S. 579 (1993), <http://supct.law.cornell.edu/supct/html/92-102.ZS.html>.

for these categories as up to 15 years for the first offence, and 5-15 years if there were prior convictions under in this chapter (Chapter 110, “sexual exploitation and other abuse of children”) or other sex offenses (Chapter 109A and 117 or state laws relating to aggravated sexual abuse, sexual abuse, or abusive sexual conduct involving a minor or ward, or the production, possession, receipt, mailing, sale, distribution, shipment, or transportation of child pornography).

The fifth category is: (a)(5) possessing. Section 2252A(b)(2) defines the penalty for this category as up to 5 years for the first offense or 2-10 years for prior convictions in this chapter (Chapter 110, “sexual exploitation and other abuse of children”) or for other sex offenses (Chapter 109A and 117 or state laws relating to aggravated sexual abuse, sexual abuse, or abusive sexual conduct involving a minor or ward, or the production, possession, receipt, mailing, sale, distribution, shipment, or transportation of child pornography).

18 USC 2252A(d) specifies “affirmative defense” against a charge of violating subsection (a)(5) if the defendant possessed less than 3 images of child pornography and “promptly and in good faith” took reasonable steps to destroy the images and reported the matter to a law enforcement agency, allowing them access to the images.

Child pornography is contraband and must be reported to law enforcement.

18 USC 2256(2) defines what is considered “sexually explicit conduct.” In page 371 of [8], JJ McLean mentions the Dost-Standards⁷, which are used to determine what constitutes “lascivious exhibition of the genitals or pubic area” under 18 USC 2256(2)(E) and were summarized in Knox⁸; these standards provide investigators with a more clearly defined list of what constitutes child pornography.

In addition, Article 263 of the New York State Consolidated Laws defines laws relating to the “sexual performance of a child.” Section 263.11 covers “possessing an obscene sexual performance by a child” and section 263.16 covers “possessing a sexual performance by a child”; both are class E felonies.

⁷ US v. Dost 636 F. Supp. 828, 831 (S.D. California, 1986).

⁸ US v. Knox, US 3rd Circuit 1994, <http://laws.findlaw.com/3rd/940734p.html>.

References

- [1] The Autopsy Forensic Browser, <http://www.sleuthkit.org/autopsy/index.php>
- [2] bmap, ftp://ftp.scyld.com/pub/forensic_computing/bmap
- [3] bmap source from a mirror site,
<http://ftp.cfu.net/mirrors/garchive.cs.uni.edu/garchive/bmap-1.0.20/bmap-1.0.20.tar.gz>
- [4] D. Brezinski and T. Killalea, "Guidelines for Evidence Collection and Archiving," Request for Comment 3227/Best Current Practice 55, February 2002,
<http://www.ietf.org/rfc/rfc3227.txt>.
- [5] Anton Chuvakin, "Linux Data Hiding and Recovery," March 10, 2002,
http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html
- [6] Computer Forensics International, "How Hard Drives Work," http://www.cf-intl.com/evidence_recovery_basics.htm.
- [7] Eoghan Casey, *Digital Evidence and Computer Crime*, San Diego: Academic Press, 2000.
- [8] Eoghan Casey (ed), *Handbook of Computer Crime Investigation*, San Diego: Academic Press, 2002.
- [9] Forensic Tools, <http://www.forinsect.de/forensics/forensics-tools.php>.
- [10] Google, <http://www.google.com>.
- [11] netcat mirror, <http://mirror.trouble-free.net/jq/noarch/interserpatches/requires/8.0>.
- [12] New York State Consolidated Laws, Penal Code, Chapter 40, Title JA156, Article 156, Offenses Involving Computers,
<http://assembly.state.ny.us/leg/?cl=82&a=35>
- [13] Red Hat, <http://redhat.com>.
- [14] Title 17 USC 506, "Criminal offenses,"
<http://www4.law.cornell.edu/uscode/17/506.html>.
- [15] Title 18 USC 2319, "Criminal infringement of a copyright,"
<http://www4.law.cornell.edu/uscode/18/2319.html>.

[16] Title 18, Part 1, Chapter 47, Section 1030, "Fraud and related activity in connection with computers," <http://www4.law.cornell.edu/uscode/18/1030.html>.

[17] Title 18 USC 2252A, "Certain activities relating to materials constituting or containing child pornography,"
<http://www4.law.cornell.edu/uscode/18/2252A.html>.

[18] US Department of Justice, "How to Report Internet-Related Crime,"
<http://cybercrime.gov/reporting.htm>.

[19] VMware, <http://www.vmware.com>.

[20] ZymaX Forensics, "Tests for Legal Viability"
<http://www.zymaxforensics.com/forensicslegal/testslegal.htm>.

© SANS Institute 2004, Author retains full rights.

Appendix A – File timeline

In this appendix, we include a timeline of the MAC times of the files on the floppy. Note that there are a number of c-time entries. It is possible that Mr. Price made extensive use of the `touch` command to alter the modify (M) and access (A) times to destroy the evidence trail. Note that the `bmap` program does *not* affect MAC times on files.

Tue Jan 28 2003 10:56:00	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 06:08:00	1024	m..	d/drwxr-xr-x	502	502	12	/mnt/floppy/John
Sat May 03 2003 06:10:00	1024	m..	d/drwxr-xr-x	502	502	14	/mnt/floppy/May03
Wed May 21 2003 06:09:00	29184	ma.	-/-rwxr-xr-x	502	502	13	/mnt/floppy/Docs/DVD-
Playing-HOWTO-html.tar	27430	ma.	-/-rwxr-xr-x	502	502	19	/mnt/floppy/Docs/Kernel-
HOWTO-html.tar.gz							
Wed May 21 2003 06:12:00	32661	ma.	-/-rwxr-xr-x	502	502	20	/mnt/floppy/Docs/MP3-
HOWTO-html.tar.gz							
Wed Jun 11 2003 09:09:00	29696	ma.	-/-rw-----	502	502	16	
/mnt/floppy/Docs/Letter.doc							
Mon Jul 14 2003 10:08:09	12288	m.c	d/drwx-----	0	0	11	/mnt/floppy/lost+found
	0	mac	-----	0	0	1	<fl-160703-jp1.dd-alive-1>
Mon Jul 14 2003 10:11:50	26843	ma.	-/-rwxr-xr-x	502	502	21	/mnt/floppy/Docs/Sound-
HOWTO-html.tar.gz							
Mon Jul 14 2003 10:12:02	56950	ma.	-/-rwxr-xr-x	502	502	22	/mnt/floppy/nc-1.10-
16.i386.rpm..rpm							
Mon Jul 14 2003 10:12:15	100430	ma.	-rwxr-xr-x	0	0	23	<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:12:48	13487	ma.	-/-rwxr-xr-x	502	502	26	
/mnt/floppy/May03/ebay300.jpg							
Mon Jul 14 2003 10:13:13	546116	m..	-rwxr-xr-x	502	502	27	<fl-160703-jp1.dd-dead-27>
Mon Jul 14 2003 10:13:52	2592	m.c	-/-rw-r--r--	0	0	28	/mnt/floppy/.~5456g.tmp
Mon Jul 14 2003 10:19:13	100430	..c	-rwxr-xr-x	0	0	23	<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:22:36	1024	m..	d/drwxr-xr-x	502	502	15	/mnt/floppy/Docs
Mon Jul 14 2003 10:24:00	487476	m..	-/-rwxr-xr-x	502	502	18	/mnt/floppy/prog
Mon Jul 14 2003 10:43:44	1024	..c	d/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 10:43:53	13487	..c	-/-rwxr-xr-x	502	502	26	
/mnt/floppy/May03/ebay300.jpg							
Mon Jul 14 2003 10:43:57	56950	..c	-/-rwxr-xr-x	502	502	22	/mnt/floppy/nc-1.10-
16.i386.rpm..rpm							
Mon Jul 14 2003 10:45:48	29184	..c	-/-rwxr-xr-x	502	502	13	/mnt/floppy/Docs/DVD-
Playing-HOWTO-html.tar							
Mon Jul 14 2003 10:46:00	27430	..c	-/-rwxr-xr-x	502	502	19	/mnt/floppy/Docs/Kernel-
HOWTO-html.tar.gz							
Mon Jul 14 2003 10:46:07	32661	..c	-/-rwxr-xr-x	502	502	20	/mnt/floppy/Docs/MP3-
HOWTO-html.tar.gz							
Mon Jul 14 2003 10:47:10	546116	.a.	-rwxr-xr-x	502	502	27	<fl-160703-jp1.dd-dead-27>
Mon Jul 14 2003 10:47:57	29696	..c	-/-rw-----	502	502	16	
/mnt/floppy/Docs/Letter.doc							
Mon Jul 14 2003 10:48:15	19456	mac	-/-rw-----	502	502	17	
/mnt/floppy/Docs/Mikemsg.doc							
Mon Jul 14 2003 10:48:53	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 10:49:25	1024	..c	d/drwxr-xr-x	502	502	12	/mnt/floppy/John
Mon Jul 14 2003 10:50:15	1024	..c	d/drwxr-xr-x	502	502	14	/mnt/floppy/May03
Wed Jul 16 2003 02:03:00	546116	..c	-rwxr-xr-x	502	502	27	<fl-160703-jp1.dd-dead-27>
Wed Jul 16 2003 02:03:13	1024	m.c	-/drwxr-xr-x	0	0	2	/mnt/floppy/John/
(deleted-realloc)							
Wed Jul 16 2003 02:05:33	487476	..c	-/-rwxr-xr-x	502	502	18	/mnt/floppy/prog
Wed Jul 16 2003 02:06:15	12288	.a.	d/drwx-----	0	0	11	/mnt/floppy/lost+found
Wed Jul 16 2003 02:09:35	1024	.a.	d/drwxr-xr-x	502	502	12	/mnt/floppy/John
Wed Jul 16 2003 02:09:49	1024	.a.	d/drwxr-xr-x	502	502	14	/mnt/floppy/May03
Wed Jul 16 2003 02:10:01	1024	.a.	d/drwxr-xr-x	502	502	15	/mnt/floppy/Docs
Wed Jul 16 2003 02:11:36	2592	.a.	-/-rw-r--r--	0	0	28	/mnt/floppy/.~5456g.tmp
Wed Jul 16 2003 02:12:39	1024	.a.	-/drwxr-xr-x	0	0	2	/mnt/floppy/John/
(deleted-realloc)							
Wed Jul 16 2003 02:12:45	487476	.a.	-/-rwxr-xr-x	502	502	18	/mnt/floppy/prog

Appendix B – Files sorted by inode

In this appendix, we list the files, sorted by inode number as generated by the command `ls -li`. The inode number is the first column in each line. In the ext2 file system, block 1 contains the bad block list, and block 2 is the root node. The first user-visible block is inode 11, which is `lost+found` which the system generates when the file system is created (via `mkfs`). The inodes for files within a directory are generally allocated in sequential order. Deleted inodes are then (re)allocated.

It is possible, for example, that inode 13 (“DVD...”) was created *after* inode 15 (“Docs”) and has a number less than the directory that contains it because 13 originally was used by a file that was deleted. Alternatively (and more likely), the file was created *before* the directory and was later moved into the directory. Normally, this would be reflected in a c-time change of the file. Note that inodes 23 and 27 are not currently in use; they were used by files that were deleted (this is reflected on the timeline in Appendix A).

This type of analysis provides another view of the relative order in which files were created which can be compared to the MAC timeline. And just like the MAC timeline data *can* be corrupted via the `touch` command, creating and deleting many junk files can create many gaps in the inode table, corrupting the information here. However, the floppy disk does not contain that many files, so it is less likely that sort of anti-forensic technique was employed.

11	drwx-----	2	root	root	12288	Jul 14	2003	/lost+found
12	drwxr-xr-x	2	502	502	1024	Feb 3	2003	/John
13	-rwxr-xr-x	1	502	502	29184	May 21	2003	/Docs/DVD-Playing-HOWTO-html.tar
14	drwxr-xr-x	2	502	502	1024	May 3	2003	/May03
15	drwxr-xr-x	2	502	502	1024	Jul 14	2003	/Docs
16	-rw-----	1	502	502	29696	Jun 11	2003	/Docs/Letter.doc
17	-rw-----	1	502	502	19456	Jul 14	2003	/Docs/Mikemsg.doc
18	-rwxr-xr-x	1	502	502	487476	Jul 14	2003	/prog
19	-rwxr-xr-x	1	502	502	27430	May 21	2003	/Docs/Kernel-HOWTO-html.tar.gz
20	-rwxr-xr-x	1	502	502	32661	May 21	2003	/Docs/MP3-HOWTO-html.tar.gz
21	-rwxr-xr-x	1	502	502	26843	Jul 14	2003	/Docs/Sound-HOWTO-html.tar.gz
22	-rwxr-xr-x	1	502	502	56950	Jul 14	2003	/nc-1.10-16.i386.rpm..rpm
24	-rwxr-xr-x	1	502	502	19088	Jan 28	2003	/John/sect-num.gif
25	-rwxr-xr-x	1	502	502	20680	Jan 28	2003	/John/sectors.gif
26	-rwxr-xr-x	1	502	502	13487	Jul 14	2003	/May03/ebay300.jpg
28	-rw-r--r--	1	root	root	2592	Jul 14	2003	/..5456g.tmp

Appendix C – tar2d2 output from tar under Linux

In this appendix, we include the output file that results from running tar2d2 on a Red Hat Linux system, using tar with the 'xf' parameters.

```
<?xml version='1.0' standalone='yes'?>
<main>
  <result>
    <name>PID 24504, run 0</name>
    <basename>/root/sansresults/blarg</basename>
    <command>/bin/tar xf /tmp/sans.tar</command>
    <file>
      <name>/firstdirectory</name>
      <MD5>0</MD5>
      <atime>1082321548</atime>
      <blksize>4096</blksize>
      <blocks>8</blocks>
      <ctime>1082321548</ctime>
      <dev>771</dev>
      <gid>0</gid>
      <ino>3745469</ino>
      <mode>16877</mode>
      <mtime>1081904847</mtime>
      <nlink>2</nlink>
      <rdev>0</rdev>
      <size>4096</size>
      <uid>0</uid>
    </file>
    <file>
      <name>/firstdirectory/readsubdirfile</name>
      <MD5>8272da861b889760891513f2477c1193</MD5>
      <atime>1082321548</atime>
      <blksize>4096</blksize>
      <blocks>8</blocks>
      <ctime>1082321548</ctime>
      <dev>771</dev>
      <gid>0</gid>
      <ino>3745470</ino>
      <mode>33188</mode>
      <mtime>1081904847</mtime>
      <nlink>1</nlink>
      <rdev>0</rdev>
      <size>37</size>
      <uid>0</uid>
    </file>
    <file>
      <name>/readfile</name>
      <MD5>bae6901663dbd94c3933acc3eca1bb54</MD5>
      <atime>1082321548</atime>
      <blksize>4096</blksize>
      <blocks>8</blocks>
      <ctime>1082321548</ctime>
      <dev>771</dev>
      <gid>0</gid>
      <ino>2829761</ino>
      <mode>33188</mode>
      <mtime>1081904704</mtime>
      <nlink>1</nlink>
      <rdev>0</rdev>
      <size>58</size>
      <uid>0</uid>
    </file>
    <file>
      <name>/writefile</name>
      <MD5>4d7bbbba20506585f048aa9af3ba67af</MD5>
      <atime>1082321548</atime>
```

```

    <blksize>4096</blksize>
    <blocks>8</blocks>
    <ctime>1082321548</ctime>
    <dev>771</dev>
    <gid>0</gid>
    <ino>2829762</ino>
    <mode>33188</mode>
    <mtime>1081904718</mtime>
    <nlink>1</nlink>
    <rdev>0</rdev>
    <size>48</size>
    <uid>0</uid>
  </file>
  <file>
    <name>/movefile</name>
    <MD5>3adf760cdf3429476e7ec542af792c25</MD5>
    <atime>1082321548</atime>
    <blksize>4096</blksize>
    <blocks>8</blocks>
    <ctime>1082321548</ctime>
    <dev>771</dev>
    <gid>0</gid>
    <ino>2829763</ino>
    <mode>33188</mode>
    <mtime>1081904742</mtime>
    <nlink>1</nlink>
    <rdev>0</rdev>
    <size>49</size>
    <uid>0</uid>
  </file>
  <file>
    <name>/chmodfile</name>
    <MD5>5c68f0cda66541fa9e59b1599d93ee68</MD5>
    <atime>1082321548</atime>
    <blksize>4096</blksize>
    <blocks>8</blocks>
    <ctime>1082321548</ctime>
    <dev>771</dev>
    <gid>0</gid>
    <ino>2829764</ino>
    <mode>33204</mode>
    <mtime>1081904774</mtime>
    <nlink>1</nlink>
    <rdev>0</rdev>
    <size>44</size>
    <uid>0</uid>
  </file>
  <file>
    <name>/secondsubdirectory</name>
    <MD5>0</MD5>
    <atime>1082321548</atime>
    <blksize>4096</blksize>
    <blocks>8</blocks>
    <ctime>1082321548</ctime>
    <dev>771</dev>
    <gid>0</gid>
    <ino>3123463</ino>
    <mode>16877</mode>
    <mtime>1081904917</mtime>
    <nlink>2</nlink>
    <rdev>0</rdev>
    <size>4096</size>
    <uid>0</uid>
  </file>
  <file>
    <name>/secondsubdirectory/movesubdirfile</name>
    <MD5>0b381b4b6340e01ec64278a9cedbbb59</MD5>
    <atime>1082321548</atime>
    <blksize>4096</blksize>
    <blocks>8</blocks>
    <ctime>1082321548</ctime>

```

```

    <dev>771</dev>
    <gid>0</gid>
    <ino>3123465</ino>
    <mode>33188</mode>
    <mtime>1081904902</mtime>
    <nlink>1</nlink>
    <rdev>0</rdev>
    <size>73</size>
    <uid>0</uid>
  </file>
  <numberoffiles>8</numberoffiles>
  <overwritten>
    <atime>8</atime>
    <blocks>8</blocks>
    <ctime>8</ctime>
    <dev>8</dev>
    <ino>8</ino>
    <mode>8</mode>
    <mtime>3</mtime>
    <nlink>2</nlink>
    <rdev>2</rdev>
    <size>2</size>
  </overwritten>
  <preserved>
    <name>8</name>
    <MD5>8</MD5>
    <blksize>8</blksize>
    <gid>8</gid>
    <mtime>5</mtime>
    <nlink>6</nlink>
    <rdev>6</rdev>
    <size>6</size>
    <uid>8</uid>
  </preserved>
  <rc>0</rc>
  <rundate>Sun Apr 18 16:52:28 2004</rundate>
  <start>1082321548</start>
  <stop>1082321548</stop>
</result>
</main>

```

Appendix D – tar2d2 output from tar under XP

In this appendix, we include the output file that results from running tar2d2 on a Windows XP Pro system, using tar with the 'xf' parameters. Note that the same data is present as in Appendix C although the tags are in a different order. Also, because of difference between Windows and Linux, some of the values returned by the `stat()` command are always zero under Windows. `atime`, `ctime`, and `mtime` are correctly returned.

```
<?xml version='1.0' standalone='yes'?>
<main>
  <result>
    <basename>C:\cygwin\home\MFP User\sans</basename>
    <rc>0</rc>
    <command>\cygwin\bin\tar.exe xf ../sans.tar</command>
    <rundate>Sun Apr 18 16:14:13 2004</rundate>
    <name>PID 2736, run 0</name>
    <numberoffiles>8</numberoffiles>
    <start>1082319253</start>
    <stop>1082319253</stop>
    <file>
      <blocks></blocks>
      <size>44</size>
      <uid>0</uid>
      <MD5>5c68f0cda66541fa9e59b1599d93ee68</MD5>
      <blksize></blksize>
      <dev>2</dev>
      <name>\chmodfile</name>
      <ino>0</ino>
      <gid>0</gid>
      <rdev>2</rdev>
      <ctime>1082247807</ctime>
      <mode>33206</mode>
      <nlink>1</nlink>
      <atime>1082319253</atime>
      <mtime>1081904774</mtime>
    </file>
    <file>
      <blocks></blocks>
      <size>0</size>
      <uid>0</uid>
      <MD5>0</MD5>
      <blksize></blksize>
      <dev>2</dev>
      <name>\firstdirectory</name>
      <ino>0</ino>
      <gid>0</gid>
      <rdev>2</rdev>
      <ctime>1082319253</ctime>
      <mode>16895</mode>
      <nlink>1</nlink>
      <atime>1082319253</atime>
      <mtime>1081904847</mtime>
    </file>
    <file>
      <blocks></blocks>
      <size>37</size>
      <uid>0</uid>
      <MD5>8272da861b889760891513f2477c1193</MD5>
      <blksize></blksize>
      <dev>2</dev>
      <name>\firstdirectory\readsubdirfile</name>
      <ino>0</ino>
      <gid>0</gid>
```



```

    <rdev>2</rdev>
    <ctime>1082319253</ctime>
    <mode>33206</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904847</mtime>
  </file>
  <file>
    <blocks></blocks>
    <size>49</size>
    <uid>0</uid>
    <MD5>3adf760cdf3429476e7ec542af792c25</MD5>
    <blksize></blksize>
    <dev>2</dev>
    <name>\movefile</name>
    <ino>0</ino>
    <gid>0</gid>
    <rdev>2</rdev>
    <ctime>1082247807</ctime>
    <mode>33206</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904742</mtime>
  </file>
  <file>
    <blocks></blocks>
    <size>58</size>
    <uid>0</uid>
    <MD5>bae6901663dbd94c3933acc3eca1bb54</MD5>
    <blksize></blksize>
    <dev>2</dev>
    <name>\readfile</name>
    <ino>0</ino>
    <gid>0</gid>
    <rdev>2</rdev>
    <ctime>1082247807</ctime>
    <mode>33206</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904704</mtime>
  </file>
  <file>
    <blocks></blocks>
    <size>0</size>
    <uid>0</uid>
    <MD5>0</MD5>
    <blksize></blksize>
    <dev>2</dev>
    <name>\secondsubdirectory</name>
    <ino>0</ino>
    <gid>0</gid>
    <rdev>2</rdev>
    <ctime>1082319253</ctime>
    <mode>16895</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904917</mtime>
  </file>
  <file>
    <blocks></blocks>
    <size>73</size>
    <uid>0</uid>
    <MD5>0b381b4b6340e01ec64278a9cedbbb59</MD5>
    <blksize></blksize>
    <dev>2</dev>
    <name>\secondsubdirectory\movesubdirfile</name>
    <ino>0</ino>
    <gid>0</gid>
    <rdev>2</rdev>
    <ctime>1082319253</ctime>

```

```

    <mode>33206</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904902</mtime>
  </file>
  <file>
    <blocks></blocks>
    <size>48</size>
    <uid>0</uid>
    <MD5>4d7bbba20506585f048aa9af3ba67af</MD5>
    <blksize></blksize>
    <dev>2</dev>
    <name>\writefile</name>
    <ino>0</ino>
    <gid>0</gid>
    <rdev>2</rdev>
    <ctime>1082247807</ctime>
    <mode>33206</mode>
    <nlink>1</nlink>
    <atime>1082319253</atime>
    <mtime>1081904718</mtime>
  </file>
  <preserved>
    <uid>8</uid>
    <size>8</size>
    <blocks>8</blocks>
    <MD5>8</MD5>
    <blksize>8</blksize>
    <name>8</name>
    <dev>8</dev>
    <ino>8</ino>
    <rdev>8</rdev>
    <gid>8</gid>
    <mode>8</mode>
    <nlink>8</nlink>
    <mtime>8</mtime>
  </preserved>
  <overwritten>
    <ctime>8</ctime>
    <atime>8</atime>
  </overwritten>
</result>
</main>

```