Global Information Assurance Certification Paper

GCFA PRACTICAL ASSIGNMENT
V 1.4

JEFF McGURK

4/28/2004

Abstract

The following paper reflects a hands-on exercise in the world of digital forensics. It is divided into 3 parts:

- Analysis of an unknown binary
- Analysis of a suspect's system
- Discussion of the legal implications of incident response

Part 1 includes both static and dynamic analyses of the provided binary file. Static analysis involves determining the true identity of the program and what its intended functions are. Dynamic analysis includes running the program and recording what changes it affects on a system.

Part 2 is the analysis of a real system found "in the wild" and suspected to have been used for producing counterfeit checks and ID cards. The analysis involved recovering deleted information, determining likelihood of a system compromise and locating checks or IDs produced using the system.

Part 3 discusses some legal issues facing network managers who discover illegal activity on the part of their users. This included statutes regarding mandatory reporting and liability as contributing offenders.

Part 1
Unknown Binary Analysis

Case Summary

On 10/27/2003, investigator Smith of CCNOU Corp. contacted me regarding pending administrative action against an employee. A recent audit of CCNOU's router logs indicated that John Price had been illegally distributing copyrighted material using company resources. Mr. Smith, leading a team of investigators, searched Mr. Price's desk on 07/16/03 at approximately 1900hrs. Among the items seized were Mr. Price's business computer, an x86-based PC running Red Hat Linux 7.3, and (1) 3.5" floppy diskette found inserted in the PC's floppy drive. Mr. Smith searched the hard drive of the seized computer and found it to be completely wiped of information. He then imaged the floppy disk found at the scene and discovered an unknown binary among the disk's files. Mr. Smith has requested that I conduct a forensic exam of the disk to attempt to determine what the unknown binary *prog* really is and to discover any information relevant to the above outlined case. The equipment used for this analysis (relevant information only) is as follows:

Dell Precision 650 Workstation
- Dual Intel Xeon 2.8GHz processors
- 2GB PC2100 DDR RAM
- Integrated Intel PRO/1000 NIC
- Integrated LSI Logic U320 SCSI Controller
  - (1) Seagate ST336753LW hdd running Windows 2000
  - (1) Seagate ST336753LW hdd running Red Hat 9
    - Vmware 4 for Linux running Red Hat 8 guest

Binary Details

Investigator Smith provided a gzip compressed dd image of the floppy seized from John Price's computer. I received the file, fl-160703-jp1.dd.gz electronically, in the file binary_v1_4.zip. Also included in the zip file was an MD5 hash of the gzip file, and an additional file Prog.md5, presumably a hash of the unidentified file *prog*.

Provided Hashes:
Image MD5: 4b680767a2aed974cec5fbcbf84cc97a
Prog MD5: 7b80d9aff486c6aa6aa3efa63cc56880

I first calculated an MD5 hash of the file binary_v1_4.zip, as shown in Figure 1, and then extracted the enclosed files issuing the command *unzip binary_v1_4.zip*. Next, I calculated the hash of fl-160703-jp1.dd.gz and received as output the hash listed above as "Image MD5" (Figure2). It bears noting that although this seems to be provided as the hash for the raw floppy image it is in fact the hash of the compressed version of that file.

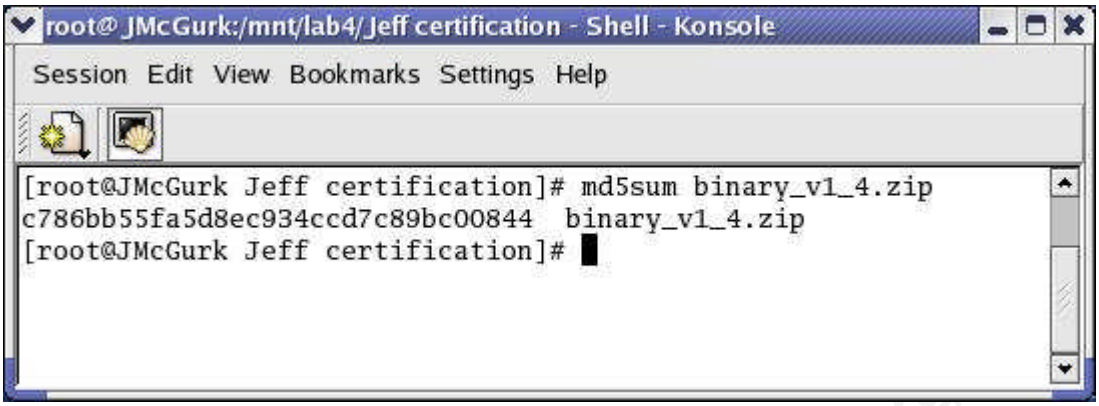

Figure 1: MD5 Hash, binary_v1_4.zip

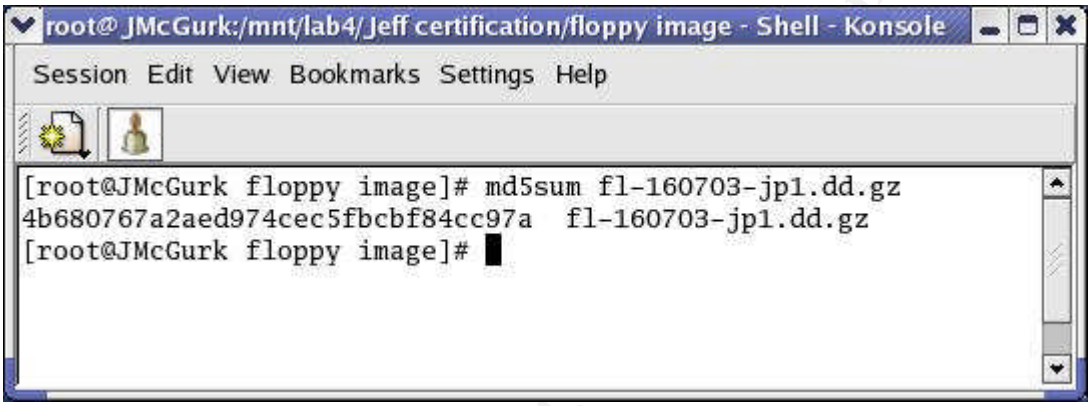

Figure 2: MD5 Hash, fl-160703-jp1.dd.gz

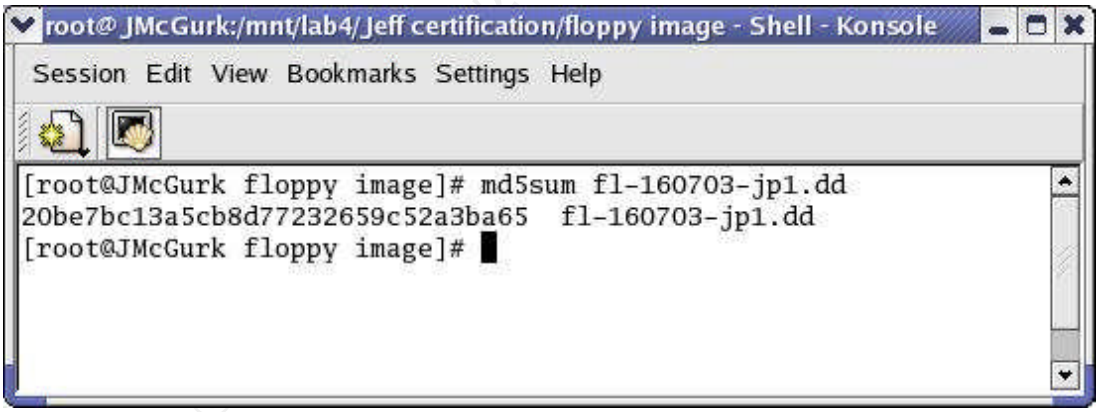

Figure 3: MD5 Hash, fl-160703-jp1.dd

My last step in preparation was to calculate an MD5 hash of the raw image file fl-160703-jp1.dd, which is shown in Figure 3. With that, I was ready to begin my analysis. I started by importing the raw image file into EnCase v4, and then reacquiring the image into the proprietary EnCase format. This allowed me to conduct further analysis later, after the Linux portion of my analysis was complete. EnCase calculated

an acquisition hash equal to that in Figure 3. This offers a level of redundancy, providing an exact duplicate backup copy of the evidence image.

I located the questioned binary *prog* within the EnCase image file and noted its properties. The discovered data was:

Modified	07/16/2003 0105
Accessed	07/16/2003 0112
Changed	07/14/2003 0924
MD5 Hash	7b80d9aff486c6aa6aa3efa63cc56880
Owner/Group	502/502
File Size (bytes)	487,476 logical / 488,448 physical

The EnCase data can be found in Figure 4.

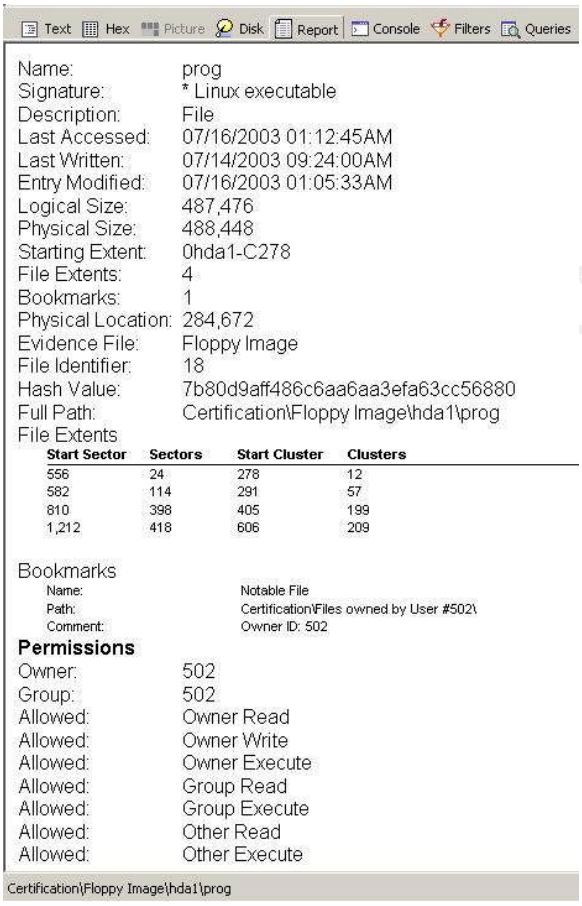

Figure 4: EnCase data for prog

Some noteworthy information is listed in the Permissions section shown in Figure 4. This file is listed as belonging to user 502, which corresponds to a Linux user name which would be stored in the /etc/passwd file. Linux begins numbering regular users at 500, reserving lower numbers for system accounts. While true that if John Price was

assigned user number 502 on his machine it would be quite convincing evidence, the fact is that anyone could have put that file there. Linux uses the same default user numbers for every installation, starting at 500 and progressing onward. A file can be placed onto a floppy using any machine, say a home system for example, and then the owner can be changed by root to be any user number – even one that doesn't exist. If the disk is then taken to a system on which that user number DOES exist, Linux will report that the file belongs to that user. Plus, since Figure 4 shows execute permissions for everyone there is no sure guarantee that user 502 was the one who executed it, if it is found to have been executed at all. These are important considerations to keep in the back of one's mind, although Ockham's Razor <http://www.merriam-webster.com/cgi-bin/dictionary?book=Dictionary&va=occam> suggests that user 502 be considered the owner of *prog* until it is shown otherwise.

Program Description

I next exported *prog* out of EnCase and returned to my virtual machine running Red Hat 8.0. I started by running the *file* command on *prog*, which performs various tests and provides information to help determine *prog*'s function. The results were:

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

The results verified that *prog* was indeed a Linux executable file. The most important information learned at this stage was that *prog* is a statically linked and stripped. Statically linked means that an executable contains any library routines needed for its execution, instead of relying on shared libraries on the host system. This allows for portability as the file contains everything needed to run on any compatible system. Stripped refers to debugging information used by programmers. The debug info can provide great insight into a programs function and is quite often removed when someone doesn't want the function discovered.

My next step was to run the *strings* command on the file, directing the output to a new file called prog.strings. The *strings* command simply searches any given file for human readable text and can be invaluable in locating help statements or other useful text within binaries. Some of the more unique strings found included:

(1) Keld Simonsen
(2) 1.0.20 (07/15/03)
(3) wipe the file from the raw device
(4) use block-list knowledge to perform special operations on files

Each phrase was searched individually in Google, and #4 resulted in a hit at <http://lwn.net/2000/0420/announce.php3> for a program called *bmap*, version 1.0.17. This hit looked promising, as the description for *bmap* matched my search term verbatim. Armed with the new information, I ran a new Google search for "linux bmap." After a few listings for kernel related information, I struck upon something intriguing indeed, a page titled "Linux Data Hiding and Recovery." The author writes "The obscure tool bmap exists to jam data in slack space, take it out and also wipe the slack

space, if needed."[1] Clicking on the word "bmap" connected to an ftp site, <ftp.scyld.com/pub/forensic_computing/bmap> which contained source code and RPM installers for the tool *bmap*, including version 1.0.20 (site no longer available, see Additional Information section.) Remembering my search term "1.0.20 (7/15/03)", I was confident I had discovered the true identity of the mysterious *prog* file.

I downloaded the source for *bmap* and, not being a heavy Linux programmer, I checked my local man pages of *gcc* for the word static. I did this by first issuing the command *man gcc* and then using */static* to search the man page. Approximately half-way down the resulting page was a section titled "Linker Options" with a "-static" option. Not knowing how to pass this -static option from the Makefile, I searched Google for "makefile static linker options." The 2nd search hit was a listserv archive message titled "Re: Static linking." The author writes that when using a pre-configured Makefile, simply "use 'LDFLAGS=-g -static'"[2] to link statically. Checking the man pages for *gcc* once again, I learned that "-g" concerns debugging information. Based on the output of the *file* command, which reported *prog* as stripped, I ignored the "-g" option and added "-static" to the makefile's already existing line of

"LDFLAGS = -L$(MFT_LIB_DIR) -lmft"

At this point I was ready to compile the *bmap* source code, and I issued the *make* command from within the source directory. Although *dvips* returned an error killing *make,* several binaries, *bclump*, *bmap, dev_builder* and *slacker* were successfully created. Issuing the *file* command on *bmap* yielded:

bmap: *ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, statically linked, not stripped*

The binary had been statically linked, but was still not stripped. The command *strip bmap* handled this, and now *file* reported:

bmap: *ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, statically linked, stripped*

I had now hypothesized that the unknown binary *prog* was in fact the data hiding tool *bmap and* downloaded the *bmap* source code. I had also compiled a stripped, statically linked *bmap* binary. The indisputable proof of *prog*'s true identity would have been if the MD5 hash value of *bmap* matched the provided *prog* hash of 7b80d9aff486c6aa6aa3efa63cc56880. Unfortunately, although not surprisingly, the hashes did not match. This was most likely due to configuration differences between the suspect's system and my own. However, all was not lost. Hashes may offer incontrovertible proof, but they are not the only way to determine that *prog* and *bmap* are one in the same. Using *strings bmap > bmap.strings* I created a text file of all printable strings within the *bmap* executable. Comparing this file to the previously created prog.strings shows a correlation that is too great to be by chance. Here's what I

[1] Chuvakin, Anton. "Linux Data Hiding and Recovery." 10 March, 2002. <http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html>(27 Oct., 2003)
[2] Mouw, Erik. "Re: Static Linking." 17 Dec., 1999. <mail.gnome.org/archives/gtk-app-devel-list/1999-December/msg00271.html> (27 Oct., 2003)

did:

sort prog.strings > prog.strings.sort
sort bmap.strings > bmap.strings.sort
comm -1 -2 prog.strings.sort bmap.strings.sort > common.strings

The 2 *sort* commands arranged all lines in each of the files into alphabetical order. Then the *comm* command compared each of the new sorted files, line by line. The "-1 -2" options told *comm* to disregard lines unique to file1 or file2, and only return those lines common to both files. The resulting file, common.strings, contained 4,100 lines! (See Figure 5.)

The command *wc -l* provides a line count of text files in Linux. It can be seen in Figure 5 that *prog* contained 4760 lines of readable text, and 4100 of them were exact matches with the executable *bmap*.

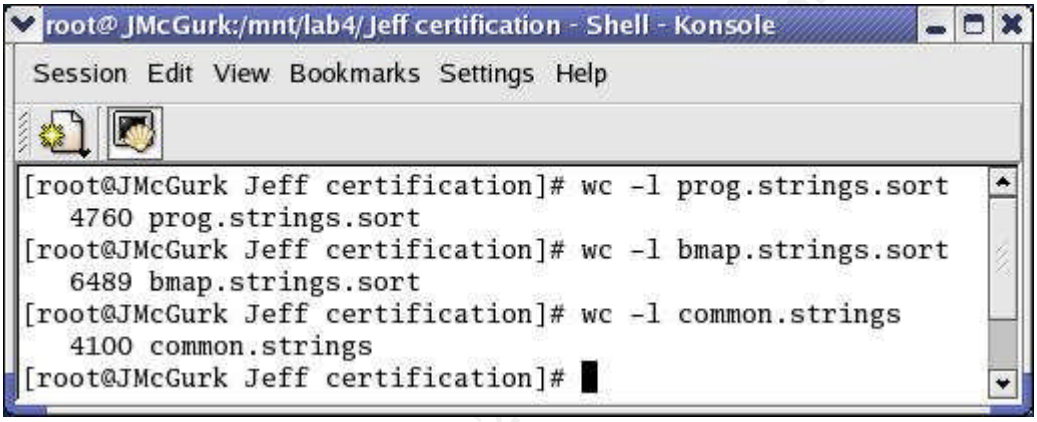

Figure 5: Common string count between prog and bmap

Some of the most significant matches were:

bmap_get_block_count
bmap_get_block_size
bmap_get_slack_block
bmap_map_block
bmap_raw_close
bmap_raw_open
wipe the file from the raw device
use block-list knowledge to perform special operations on files

As you can see from the above list, the word bmap appeared multiple times within the file *prog*. This supports the belief that *prog* and *bmap* are one in the same.

Forensic Details

Since analysis was being performed on a virtual machine, with networking disabled, contamination was not a concern. I was able to run various instances of prog and bmap and began by comparing the help statements by issuing the "--help" flag.

```
[root@JMcGurk bmap-1.0.20]# ./bmap --help
bmap:1.0.20 (11/04/03) newt@scyld.com
Usage: bmap [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files
--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  map  list sector numbers
  carve  extract a copy from the raw device
  slack  display data in slack space
  putslack  place data into slack
  wipeslack  wipe slack
  checkslack  test for slack (returns 0 if file has slack)
  slackbytes  print number of slack bytes available
  wipe  wipe the file from the raw device
  frag  display fragmentation information for the file
  checkfrag  test for fragmentation (returns 0 if file is fragmented)
--outfile <filename> write output to ...
--label useless bogus option
--name  useless bogus option
--verbose       be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit>
  logging threshold ...
--target <filename> operate on ...


[root@JMcGurk binary]# ./prog --help
prog:1.0.20 (07/15/03) newt
Usage: prog [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help  display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  m  list sector numbers
  c  extract a copy from the raw device
  s  display data
  p  place data
  w  wipe
  chk  test (returns 0 if exist)
  sb  print number of bytes available
```

```
  wipe  wipe the file from the raw device
  frag  display fragmentation information for the file
  checkfrag  test for fragmentation (returns 0 if file is fragmented)
--outfile <filename> write output to ...
--label useless bogus option
--name  useless bogus option
--verbose       be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit>
  logging threshold ...
--target <filename> operate on ...
```

Notice the striking similarities; they're almost identical, affirming again that these are in fact the same program. The most important descriptions listed for *prog*, as far as this investigation is concerned, were "place data", "wipe" and "wipe the file from the raw device." What this means is that *prog* can be used to insert data into file slack, wipe file slack, and wipe entire files from the system. Given the fact that the suspect's system was wiped of all files, there is a good chance that *prog* was used to do it. But, as the main intention of *bmap* is to deal with slack space, I decided to look there first. In the next section I will discuss what was found in slack space.

Continuing my work on the executable *prog,* I began a dynamic analysis to observe what actions the program takes when run. From the "Unix Forensics and Incident Response CD v1.9" provided by SANS Institute I ran *netstat* and *lsof* commands and routed the output to files netstat.before and lsof.before.

The *netstat* command lists the current network connections for a system along with the process ID and name of any process that owns the connection (such as an ftp server, etc.)

Running *lsof* provides a list of all currently open files along with what process is holding them open.

I also ran *ps -aux* from the host system (*ps* is not provided on the SANS CD) with an output file ps.before. *Ps* is a very common Linux/Unix command for listing active processes. I issued the -a flag to list all processes, -u to include user information, and –x to include processes not associated with a terminal.

I then ran *prog* with the -w (wipe) flag on a temporary file in /tmp. I used the wipe flag because it is possible that the suspect used *prog* to wipe the hard drive of his work computer and, if so, then *prog -w* was likely run on that system.

After running *prog*, I once again checked my system state by running *netstat*, *lsof*, and *ps*. I also ran *mac_robber*, from the SANS CD, to gather modified, accessed, and changed times of all files on the system. I used *mactime*, once again from the CD provided by SANS Institute, to parse the *mac_robber* data and send output to mactime.after. @Stake maintains both *mactime* and *mac_robber* and they can be found at <http://www.atstake.com/research/tools/forensic/>.

Next, I used the Linux *diff* command to compare netstat.before, lsof.before and ps.before with netstat.after, lsof.after and ps.after. The *diff* command compares to files,

line by line, for differences. If differences are found the table, as shown in Figure 6, indicates the line where the difference occurred and displays the line from each file for user comparison.

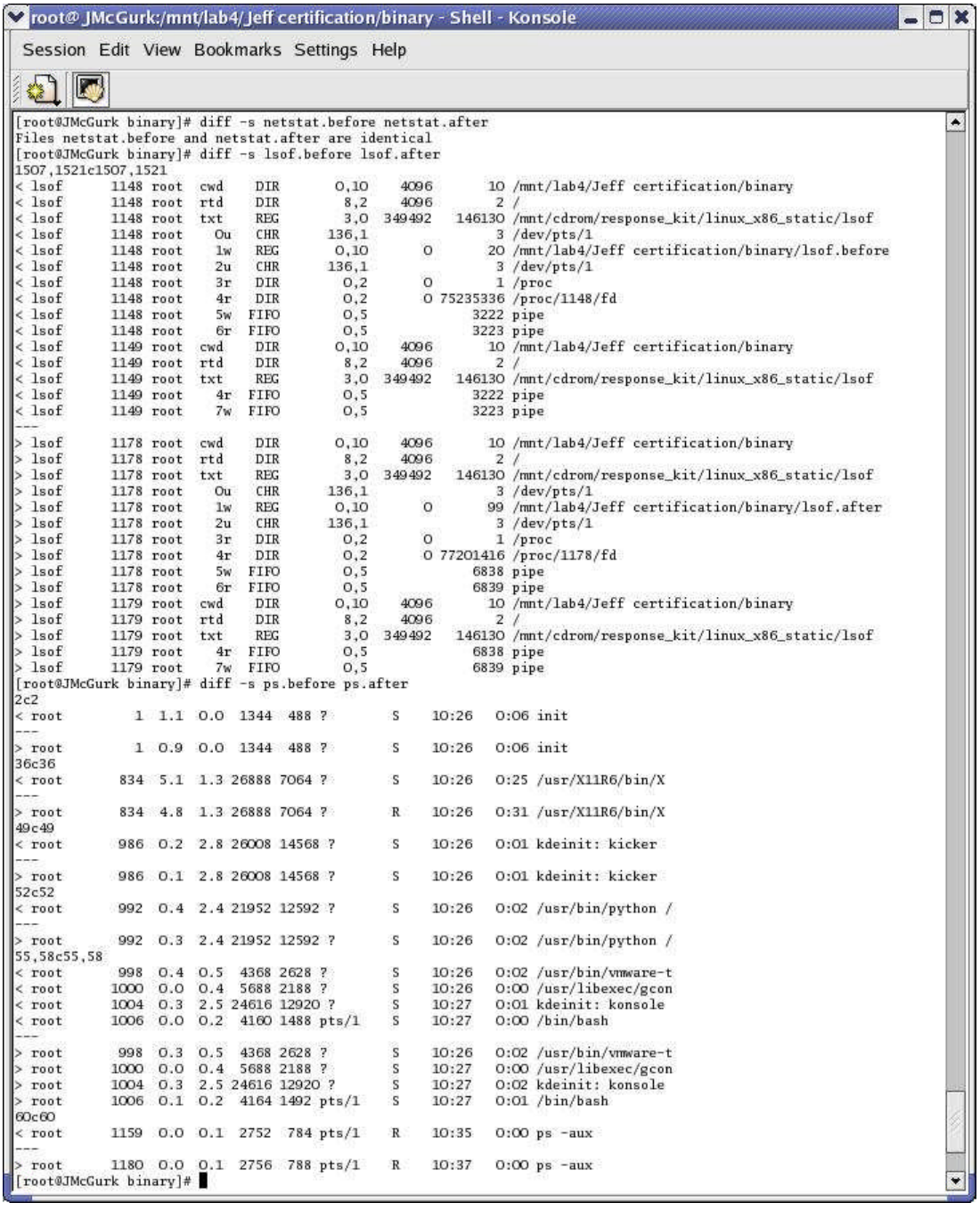

Figure 6: *netstat*, *lsof* and *ps*, compared before and after running prog

Figure 6 shows that netstat.before and netstat.after are identical. This means that no new network sockets have been opened as a result of running *prog*.

The next command shown in Figure 6 compares *lsof* output before and after running *prog.* The only differences indicated by *diff* are for *lsof* itself. This was expected because the two instances of *lsof* had different process ID numbers. Other than *lsof*, no new process was holding files open that wasn't already doing so before *prog* was run. The final section of Figure 6 shows the differences between ps.before and ps.after. What I was looking for here was new processes that were not present before *prog* was executed. The majority of the differences listed deal with the same process. For example, difference number 1 is for process 1, the init process. The process was active both before and after *prog*, but the resource percentage had changed. Before *prog* the init process was using 1.1% of the CPU cycles, afterwards it was using 0.9%.

These are not the types of differences that concerned me; I was looking more for something similar to the last difference, indicated by the “60c60” header. This shows that the highest active process before *prog* was run was 1159, but then after *prog* was run the highest process was 1180. I needed to be sure that *prog* didn't spawn a rogue process that was waiting to do damage to the system. Luckily process 1180 was merely the *ps -aux* command that I issued, not anything to be concerned with.

Mon Nov 17 2003 12:24:04	626188	.a.	-rwxr-xr-x	0	0	207294	/bin/bash
	116264	.a.	-rwxr-xr-x	0	0	207386	/bin/grep
	4	.a.	lrwxrwxrwx	0	0	207296	/bin/sh
	67725	.a.	-rw-r--r--	0	0	226693	/etc/ld.so.cache
	280	.a.	-rw-r--r--	0	0	227536	/etc/mtab
	1750	.a.	-rw-r--r--	0	0	223175	/etc/nsswitch.conf
	1389	.a.	-rw-r--r--	0	0	226943	/etc/passwd
	1395734	.a.	-rwxr-xr-x	0	0	318794	/lib/i686/libc-2.2.93.so
	14	.a.	lrwxrwxrwx	0	0	318795	/lib/i686/libc.so.6
	87341	.a.	-rwxr-xr-x	0	0	175384	/lib/ld-2.2.93.so
	12	.a.	lrwxrwxrwx	0	0	175385	/lib/ld-linux.so.2
	11314	.a.	-rwxr-xr-x	0	0	175397	/lib/libdl-2.2.93.so
	15	.a.	lrwxrwxrwx	0	0	175398	/lib/libdl.so.2
	90444	.a.	-rwxr-xr-x	0	0	175401	/lib/libnsl-2.2.93.so
	16	.a.	lrwxrwxrwx	0	0	175402	/lib/libnsl.so.1
	42657	.a.	-rwxr-xr-x	0	0	175417	/lib/libnss_files-2.2.93.so
	22	.a.	lrwxrwxrwx	0	0	175419	/lib/libnss_files.so.2
	50024	.a.	-rwxr-xr-x	0	0	175425	/lib/libnss_nisplus-2.2.93.so
	24	.a.	lrwxrwxrwx	0	0	175426	/lib/libnss_nisplus.so.2
	19	.a.	lrwxrwxrwx	0	0	175999	/lib/libtermcap.so.2
	11696	.a.	-rwxr-xr-x	0	0	175998	/lib/libtermcap.so.2.0.8
	64153	.a.	-rwxr-xr-x	0	0	207285	/sbin/ifconfig
	20866	.a.	-rw-r--r--	0	0	382673	/usr/lib/gconv/gconv-modules.cache
	1830272	.a.	-rw-r--r--	0	0	63811	/usr/lib/locale/locale-archive
	42657	.a.	-rwxr-xr-x	0	0	175417	/var/spool/postfix/lib/libnss_files-2.2.93.so
	50024	.a.	-rwxr-xr-x	0	0	175425	/var/spool/postfix/lib/libnss_nisplus-2.2.93.so
Mon 11/17/2003 12:24:07	0	ma.	crw-rw-rw-	0	0	65778	/dev/ptmx
	0	ma.	crw-------	0	5	3	/dev/pts/1
	0	.a.	crw--w----	0	0	70458	/dev/tty7
Mon 11/17/2003 12:24:10	15986688	mac	-rwxr-xr-x	0	0	61	/mnt/lab4/Jeff certification/binary/macrobber.after

Figure 7: Files affected by prog and bmap

It seemed, from the results of *netstat*, *lsof* and *ps*, that *prog* did not attempt to compromise the system with any sort of backdoor. To ensure that nothing differed from my own clean copy, I ran *bmap -wipe* on a second temporary file. I then ran *mac_robber* and *mactime* once again, to gather file MAC times as changed by *bmap*. I compared my two mactime.after files for any unnecessary file access or modification on the part of *prog*. I first loaded each file into a spreadsheet, and then alphabetically sorted all files that occurred at exactly the same time. This made it easier to compare

between the two files. Both lists had precisely the same files, in exactly the same grouping. Figure 7 shows the files affected by *prog*. The list for *bmap* is not shown since it is nearly identical, with only the times being different between the two. Since the (somewhat) trusted *bmap* affected all of the same files that *prog* did, I felt confident that *prog* contained no malware.

Two interesting observations were made while experimenting with *prog/bmap.* The first deals with the modified and accessed times of files which *prog* is used to inject information into or read/wipe information out of. By its very design, *prog* acts not on files, but on disk blocks allocated to those files. This means that the file system is not invoked in the traditional way, and MAC times of any files used to contain hidden data will not be affected.

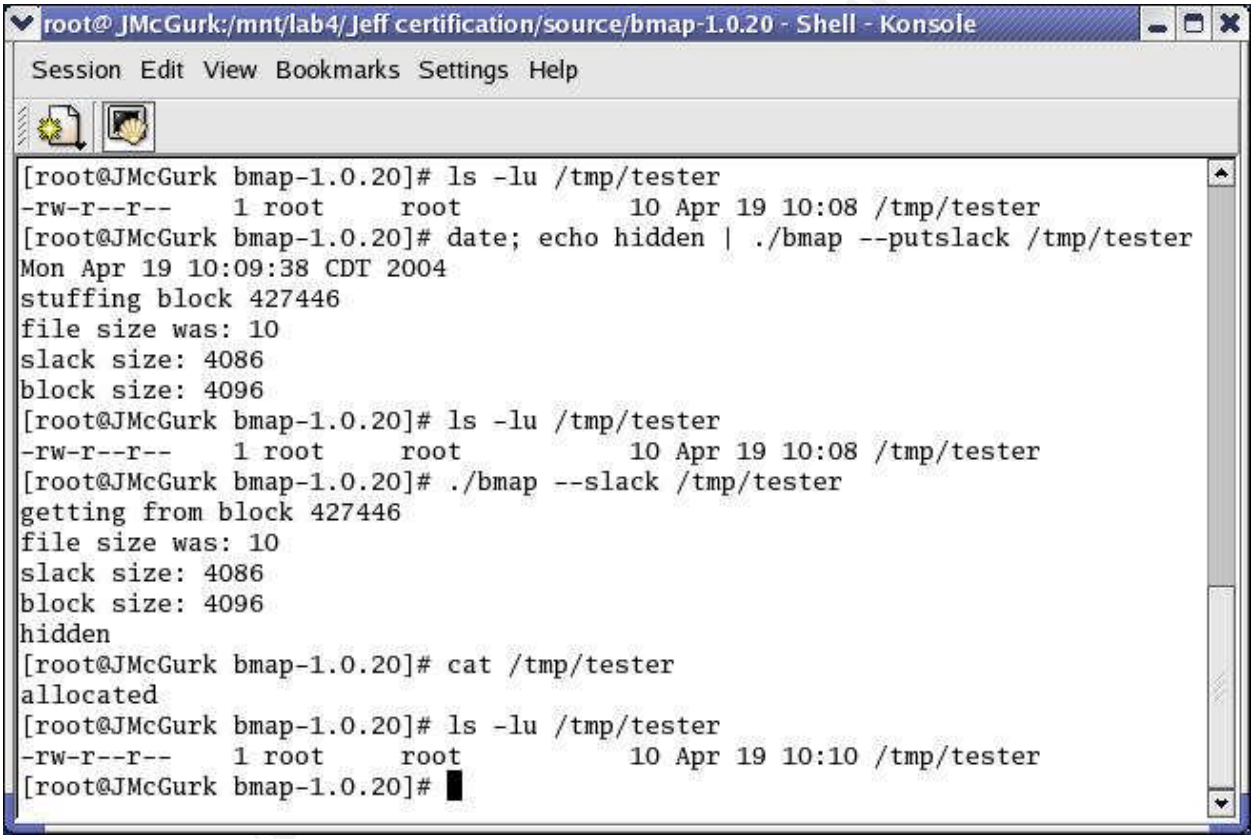

Figure 8: prog/bmap does not affect file access time

In the above illustration I began by creating a file called tester in the /tmp directory. The *ls –lu* shows that the file was accessed at 1008 (-l gives a long list and –u tells *ls* to sort by access time.) Next you can see that at 1009 I used bmap to insert the word “hidden” into the slack space of /tmp/tester. Another *ls –lu* indicates that the access time has not been updated by the action even though, as the following command shows, the word “hidden” was successfully injected. I then accessed the file using the *cat* command, which types out the file’s contents to standard out, and the final *ls –lu* shows that this now has updated the access time.

The second observation regarding prog/bmap is a little more involved than the first. While running tests on *prog* and *bmap* I ran *strace* while using the -wipeslack flag. The following is an excerpt of the results.

```
write(2, "stuffing block 427826\n", 22stuffing block 427826) = 22
write(2, "file size was: 15\n", 18file size was: 15)     = 18
write(2, "slack size: 4081\n", 17slack size: 4081)      = 17
write(2, "block size: 4096\n", 17block size: 4096)      = 17
_llseek(4, 1752375311, [1752375311], SEEK_SET) = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4081) = 4081
write(2, "write error\n", 12write error)          = 12
_llseek(4, 1752375311, [1752375311], SEEK_SET) = 0
write(4, "\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377"..., 4081) = 4081
write(2, "write error\n", 12write error)          = 12
_llseek(4, 1752375311, [1752375311], SEEK_SET) = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4081) = 4081
write(2, "write error\n", 12write error)          = 12
close(3)                                = 0
close(4)                                = 0
exit_group(0)                           = ?
```

Line 1 indicates which block the file slack is located in. The seek value in line 5 created some concern, because the block reported in line 1, 427826, multiplied by the block size of 4096 equals the exact 1752375311 bytes being indicated within the seek command. But the slack space doesn't start for another 15 bytes into the block (file size in line 2). This means that *prog/bmap* is seeking to the start of the block and not the start of slack. I thought, incorrectly as it turns out, that this indicated some bug in the functionality of *prog/bmap*. The write errors reported after each write attempt reinforced this suspicion. I tried the *prog* command again, this time with the -wipe flag to wipe an entire file. Figure 9 shows that this did not go as planned either.

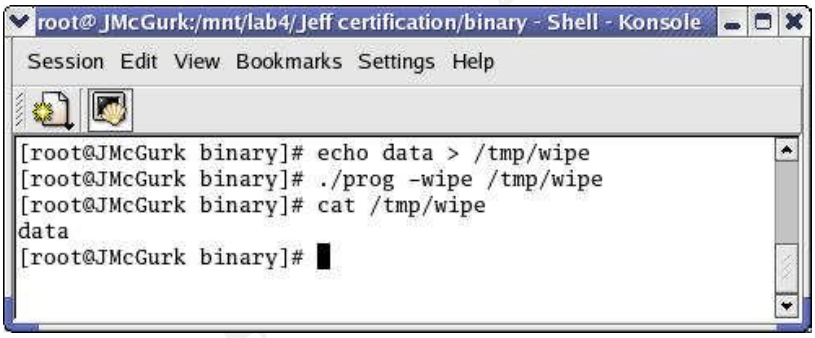

Figure 9: *prog -wipe* seems to have failed

The data from the file "/tmp/wipe" seemed to be fully in tact, not wiped at all! I thought for sure there must be a failure somewhere in the execution of the program. I wrote an email to Anton Chuvakin, whose writing cited above had first introduced me to *prog*'s intended uses, asking for any insight. Dr. Chuvakin's response was that "the tools is old (2000) and might not like the newest Linux (I assume that is what you have) file system

changes (at least that is what I think off the top of my head)..."[3] This suggestion seemed to make sense immediately. I thought back to the system differences that were the most likely cause of the differing hashes for *prog* and my own *bmap*.

I decided to attempt to determine what operating system was used to compile the file *prog*. Consulting my *strings* output, I found the strings "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)" and "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-113)." I downloaded the ISO files for Red Hat 7.3 and created a new virtual machine running RH7.3. Checking *gcc* I found that the default version was 2.96-110. I updated *gcc* to 2.96-112 and compiled a new *bmap* executable. This one still provided the write errors for -wipeslack and did not appear to wipe the entire file.

My decision was that the issue would require more research than its relevance to the case would allow. I felt I had gathered enough information about *prog* and the errors would have to go unsolved. And then, somehow, I stumbled on the answer. I returned to the virtual machine running Red Hat 8.0 and I noticed that my test file "wipe" had been cleanly wiped after all. I was a bit puzzled at first, but further tests confirmed that *prog* and *bmap* had been functioning properly the whole time. The following *script* log shows that -wipeslack functioned as intended. The write errors that I observed were most likely being misreported, and were probably just an indication that *bmap* had reached the end of the block and could not proceed any further. The *dd* tool gives a similar error when it reaches the end of a device. I decided that like *dd* the *prog* error could be safely ignored. The -wipe function was a bit more interesting. It too worked 100% as intended, but when asked to *cat* the contents of the file, Linux was erroneously reporting that it had failed.

```
Script started on Fri Jan 16 13:06:51 2004
#]0;root@localhost:/mnt/lab4/Jeff certification/binary[root@localhost binary]# echo DATA > /tmp/wipe
[root@localhost binary]# echo HIDDEN | ./prog -p /tmp/wipe
stuffing block 449389
file size was: 5
slack size: 4091
block size: 4096
[root@localhost binary]# dd bs=4096 skip=449389 count=1 if=/dev/sda2 | xxd | head
1+0 records in
1+0 records out
0000000: 4441 5441 0a48 4944 4445 4e0a 0000 0000  DATA.HIDDEN.....
0000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
[root@localhost binary]# ./prog -w /tmp/wipe
stuffing block 449389
file size was: 5
```

[3] Chuvakin, Anton. <anton@netForensics.com> Re: question regarding bmap. 01/13/2004. Email to Jeff McGurk

```
slack size: 4091
block size: 4096
write error
write error
write error
[root@localhost binary]# dd bs=4096 skip=449389 count=1 if=/dev/sda2 | xxd | head
1+0 records in
1+0 records out
0000000: 4441 5441 0a00 0000 0000 0000 0000 0000  DATA............
0000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
[root@localhost binary]# ./prog -wipe /tmp/wipe
[root@localhost binary]# dd bs=4096 skip=449389 count=1 if=/dev/sda2 | xxd | head
1+0 records in
1+0 records out
0000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
[root@localhost binary]# cat /tmp/wipe
DATA
[root@localhost binary]# exit
Script done on Fri Jan 16 13:10:44 2004
```

The above log shows how I placed the word “data” into the body of the file “wipe” and used *prog* to inject the word “hidden” into the slack space. I then used the -w, or -wipeslack flag to remove “hidden” from the slack space, and the use of *dd* piped through *xxd* shows that it worked. Next I used the -wipe flag to wipe the entire file and *dd* | *xxd* again shows success. The strange thing is that *cat*, after we've already seen the disk block is empty, still reports the word “data.” There must be some type of cache that *cat* is pulling from, because “data” simply doesn't exist on the disk. I even tried the *sync* command to flush the disk buffers, but this didn't work either. A reboot is the only way I found to get an accurate reading that the file was indeed wiped.

What this means for this investigation is that John Price did not use *prog* to wipe his hard drive. *Prog* has the ability to wipe the contents of files, but a simple *ls* shows that the directory entry is left in tact. If Mr. Price's drive was completely void of information he probably ran a *dd if=/dev/zero* type command to overwrite the entire device. It is more likely that the reason Mr. Price used this tool was to hide data in the slack space of one or more files.

Case Information

With *prog* fully analyzed, and its purpose of slack space manipulation discovered, I moved on to exploring the floppy disk image for signs of its use. It is my opinion based on observing *prog* that for the CCNOU investigators to determine that it had been run on any particular system would be quite difficult. *Prog* can place and retrieve data to and from slack space without altering MAC times at all. Plus, *prog* is a finite, run-once application that does not stay resident after a user finishes a task, and it does not spawn any sub-processes that can be looked for after the fact. Therefore, the only way to get an idea of whether or not *prog* was run would be to check slack space of all files for signs of suspicious data. One could use *prog* (or *bmap*) itself in a script to accomplish this by dumping all slack to a new file and then manually checking for data of interest. Also, the bmap source includes a tool called *slacker* that manipulates the slack of an entire directory tree, which can be used to export all slack space. In analyzing the floppy seized by investigator Smith, I used EnCase to view slack in a similar fashion.

I returned to Windows/EnCase and viewed the complete file structure in one table. Including some metadata, this gave a list of 36 files. Using the EnCase hex view I went through each file alphabetically, observing the slack at the end of each file, looking for data that may have been hidden there by *prog*. The last file in the list, \Docs\Sound-HOWTO-html.tar.gz was the only file to contain any value other than 0x00 in slack space (Figure 10).

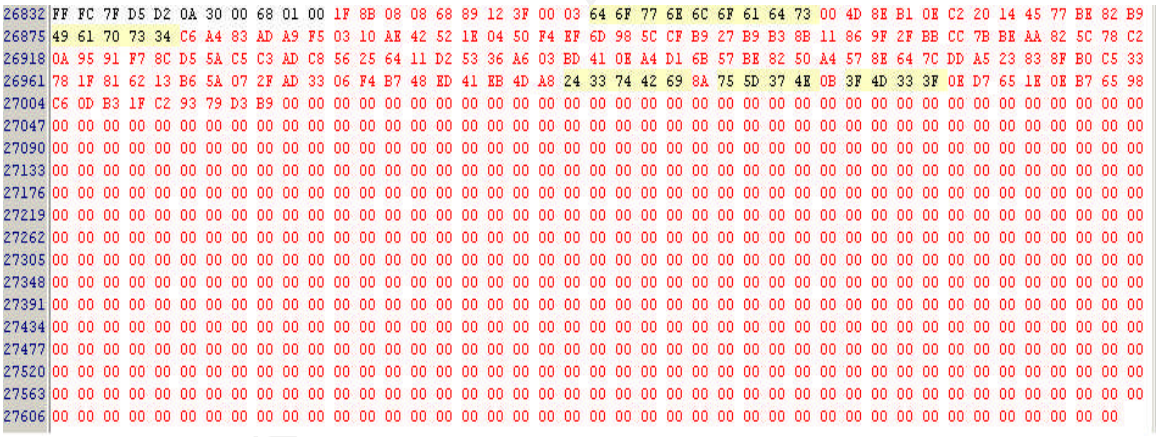

Figure 10: File slack, \Docs\Sound-HOWTO-html.tar.gz

I recognized the value 0x1F8B0808 as a header for gzip compression, and exported the data to a file for closer analysis. According to RFC1952 <http://www.faqs.org/rfcs/rfc1952.html>, the final 4 bytes of a gzipped data stream is the original uncompressed file size. The value 0xB9 at the end of the data stream could be the last byte of the file, but that would make the original size 3117644179. Barring some decompression bomb, this seems rather unlikely. Progressively moving outward, possible sizes are 12178297, 47571 and 185 bytes. Given that the amount of non-zero data present is 170 bytes, I decided to start with a value of 185. I exported the first 173 bytes of file slack, including 3 trailing 0x00 bytes and discarding the rest. I then ran the

file command on the resulting file and the response was:

```
method  crc     date  time  compressed   uncompressed  ratio uncompressed_name
defla  d37993c2 Jul 14 05:43     173          185       21.6%     downloads
```

This confirmed that the data was indeed gzip compressed, and reported that the original name was “downloads.” The value 0x646F776E6C6F616473 in Figure 10 is the hex representation of the word “downloads,” this is where *file* gets that information.

My next step was to uncompress and see exactly what “downloads” was. I issued *gzip -dN* and extracted “downloads” to the working directory. The command *ls -l* listed the following for “downloads”:

```
-rwxr-xr-x    1 root     root          185 Jul 14  2003 downloads
```

I saw that “downloads” is 185 bytes, as expected, and the date matched what *gzip* reported. Combining the two date fields, I determined that the file was last modified 07/14/2003 at 1043 GMT or 0543 Central Daylight Time.

Finally, a check of the CRC value showed that “downloads” had been correctly extracted to the same data that went into the hidden gzipped file. To check this, I used Theo Van Dinter's perl script, found at <http://www.kluge.net/~felicity/ppt/cksum> with the “-o 3” flag. The help statement for the script tells that -o “3 is CRC32r (reverse CRC32) displayed in hex.” The output was simply:

```
D37993C2 185 downloads
```

I was finally ready to look into “downloads” and see what it was that someone felt they needed to hide. I issued the *cat downloads* command and received:

```
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***
```

It seems that someone was hiding their list of illegal MP3 websites. This is definitely good circumstantial evidence against the suspect, but not concrete. The file *prog* has world execute permissions, so anyone can run it to hide data. The data hidden in the slack of Sound-HOWTO-html.tar.gz has no user data or permissions associated with it, because the file system has no knowledge of its existence. That means that determining who hid the data there is virtually impossible. Furthermore, the gzip standard, as defined in RFC1952, does not provide for storing user/permission settings, so there's no help in the data either.

However, all is not lost. Investigator Smith obviously had good evidence that John Price was involved in illegal activities. The websites listed above should help when combined

with other evidence, such as router logs. Also, Mr. Price's claim that the disk was not his to begin with can quickly be debunked.

Except for one temporary file ".~5456g.tmp" and the "lost+found" directory, all files contained within the floppy disk image belong to the previously mentioned user 502. It will be advised that investigator Smith obtain the user number assigned to Mr. Price, and with a little luck it will be 502.

If records of assigned user numbers are not available, there are also 2 Microsoft Word documents on the disk that appear to belong to John Price. "Letter.doc" is a MS Word letter template that does not appear to have been modified. But a look at the file properties told me everything I needed to know (see Figure 11). The 2nd file, "Mikemsg.doc", is also listed as belonging to John Price (Figure 11), but unlike "Letter.doc" this file has been edited to contain some useful information.

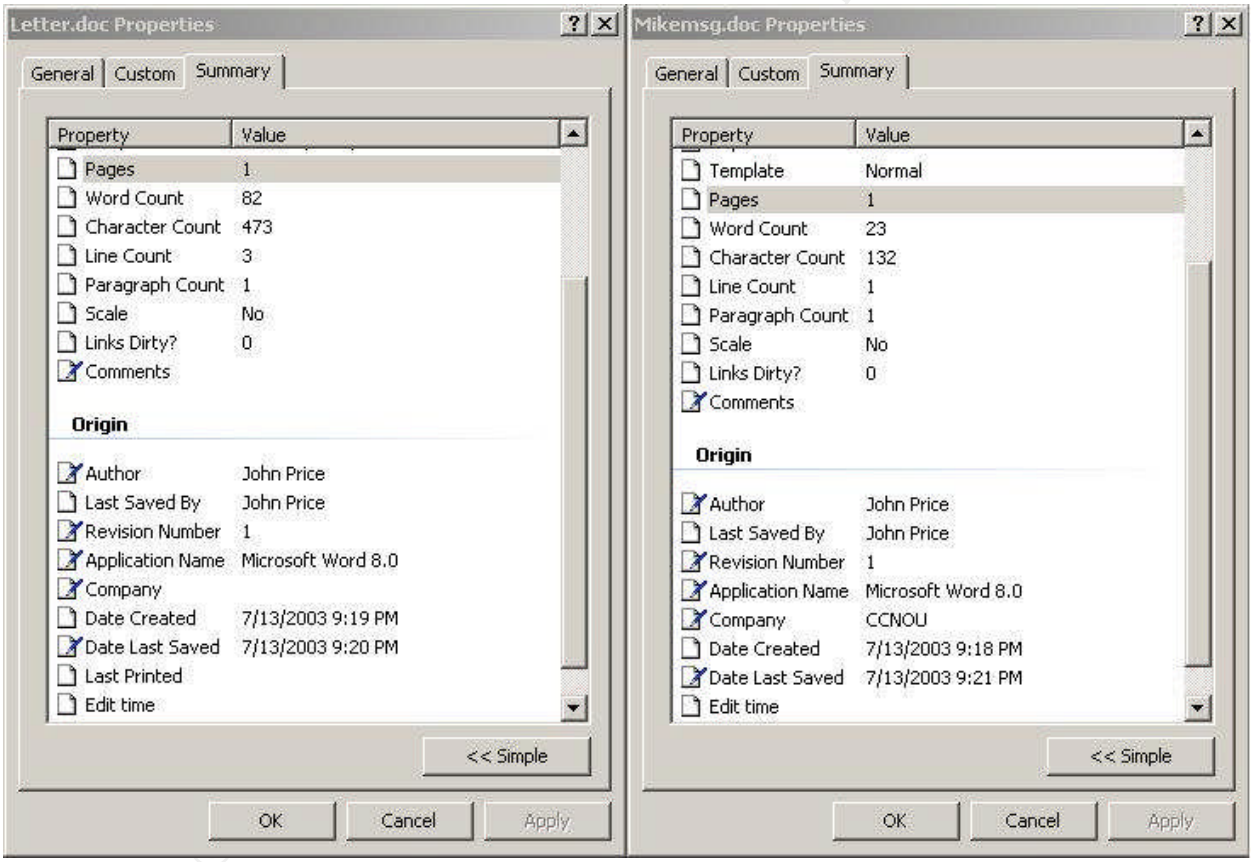

Figure 11: File properties of Letter.doc and Mikemsg.doc

The body of Mikemsg.doc is a short note:

> 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

The note is signed “JP,” the MS Word metadata lists John Price as author, and the Linux user data has the file belonging to user 502. Combined with the websites listed in the hidden “downloads” file, this note suggests that Mr. Price was working with someone named Mike to distribute illegally copied music files, in MP3 format.

The disk also contains a file “nc-1.10-16.i386.rpm..rpm”, and Linux HowTo's regarding DVD's, sound, MP3's and kernel tuning.

“Nc-1.10-16.i386.rpm..rpm” contains the program *nc* (netcat). Netcat is used for transferring data over a network (including the internet.) Any data can be sent with netcat over any port number, making it hard to detect its transfer. It is possible that Price used netcat in the alleged distribution of copyrighted material.

The Linux HowTo’s are instructional documents on system configuration and fine tuning for a particular purpose. The documents found on the floppy image relate to multimedia subjects and tweaking the kernel for specific uses.

All of these files, when taken together, suggest that Price was interested in configuring Linux for multimedia and MP3 creation and also had the capability to send files to any other network-attached computer using any predetermined port number. This isn't condemning evidence, but certainly suggests that investigator Smith has the right suspect. Network logs should be checked for any large scale activity from Mr. Price's assigned IP address to any single or small group of destination IP’s.

Beyond that, given Price’s successful wipe of the hard drive before being confronted, the best bet is to overwhelm him with the circumstantial evidence collected. The floppy disk seized by investigator Smith certainly seems to confirm that John Price was involved in distributing copyrighted material, though it does not contain “the smoking gun.” The file *prog* is an executable designed to hide data in the slack space of other files. It is known that *prog* was used to hide such data on at least one occasion. However, it is not known who used the program to hide the data since *prog* is executable by anyone. It is also not discernable from the floppy which system the program was run from when it was used. If IT can confirm that John Price’s computer had been running Red Hat 7.3 then it could be assumed under the circumstances that *prog* was compiled on that system. And if Price was user 502 then that goes a long way to refuting his denial attempts. With the abundance of circumstantial evidence, Mr. Price may be persuaded to confess his actions and put the whole situation behind him.

Legal Implications

Again, given the lack of discernable footprints, determining after the fact whether *prog/bmap* had been run on a system would prove extremely difficult, but if it had the question then becomes, “Now what?” It does not appear from the results of my analysis that prog itself or its possession, use or attempted use violates any laws of this country or state. In fact, I can envision a very effective use of *bmap* in the world of forensics for retrieving information from the slack space of files. Much like the various tools and methods employed by police in their duties, e.g. handguns, it is the ultimate use of

bmap that determines the legality. In the case of Mr. Price the suspected use was to facilitate the infringing of multiple copyrights, an issue that will be discussed in part 3 of this paper.

Although not illegal, the use or suspected use of *bmap* may violate an acceptable use policy (AUP) governing an organization's computer resources. This depends greatly on exactly how the policy is written but some points to consider are:

1. Unauthorized code.
It is a good idea to state in any AUP that users are allowed to run only software that has been approved for business purposes. It should also not be assumed, but explicitly stated that applications and code that are not on the list are not to be run under any circumstances. CCNOU should review its AUP for such a clause, and verify that John Price signed it.

2. Business use only
The AUP should stipulate that use of computing resources is authorized for legitimate purposes only, and all other uses are expressly prohibited. The use of prog will most likely not qualify as legitimate business use, and its use to distribute copyrighted material will certainly not qualify.

3. Tampering/Destruction of Data
Another clause to be included in any AUP regards the intentional destruction of data or tampering with any IT resources. This is an obvious act to prohibit, but again it should not be assumed. Specific penalties should be laid out and a list of example violations may be included, with the standard "but not limited to" clause.

Interview Questions

1. I want to thank you for your patience and cooperation through this whole thing, management is really riding my ass to get this thing cleared up. I think they're concerned about the RIAA suing them or something. It all seems a little excessive to me, but I've got my job to do. Is there anything you want to tell me before we get started?

Setting the suspect's mind at ease right up front may cause him to let his guard down as the interview progresses. He needs to know that the interviewer is on his side, and that management is to blame. Letting him know that the interviewer doesn't want the situation to drag out any longer than he does will set the tone for the remainder of the questioning, "Just a few quick points to clear up and we're out of here." Humans are inherently bad keepers of secrets. Perhaps Mr. Price will be compelled to share everything and further questions will be unnecessary.

2. Ok then, just a few quick questions. First, we know that the disk found in your computer belongs to you. It has your name all over it. And we found your letter to Mike. What can you tell me about him?

This first point must be delivered with unwavering confidence. The interviewer *knows* that the disk is his, this isn't a question. He's just mentioning it as an aside. By building a foundation of facts the interviewer is making the suspect think that everything he says is known fact. Mr. Price is given some leeway in answering this question, but his reaction must be monitored closely. This is where he decides to be honest or continue denying everything. Mike's involvement is irrelevant, because it doesn't excused Price for his actions. But Price may see this as his ticket out and blame the whole thing on Mike. The interviewer should carefully monitor the suspect's demeanor and body language, looking for signs of lying. If the suspect's answer doesn't provide any help, a follow up may lead him where we want him to go.

3. Well it seems to me that Mike was really the one breaking the law. You were just a small time accomplice. Maybe you made a few bucks; I don't see any harm there.

Again the interviewer appears to be sympathetic, to relate to the suspect's situation. But the real opportunity here is another test of human nature. Some egos just can't stand to be called "small time" and they refuse to let someone else take the credit for their work. Mr. Price may insist on refuting the claim that Mike was in charge and admit to being the brains of the operation. Or, he may go along and claim to be an unfairly persecuted patsy, forced to take the fall for Mike. Either way he will need to admit to actions that violate acceptable use of CCNOU computers.

4. That *prog* was a cool little program, I had fun playing with that. I assume you downloaded that from scyld.com?

Nothing too intimidating here, the interviewer just wants to keep the suspect talking openly. Do not give him time to stop and think. Make him react on his toes, not with some rehearsed speech. Hopefully the suspect feels comfortable enough at this point to answer honestly to questions that seem non-threatening. This sets him up for the next question.

5. It was pretty cool how you were able to hide those websites in the slack space of the SOUND-HowTo.

Once more the interviewer is on the suspect's side, playing to his ego. The undisputed fact approach along with the friendly tone set by the interviewer should at this point avoid the "I don't know what you're talking about" response. With luck Mr. Price will admit to hiding the data and the interviewer can use it for one last set-up.

6. Unfortunately it gave us some strings with which to search your hard drive. I don't know how familiar you are with computer forensics, but information is very difficult to get rid of. We were able to recover the "downloads" file with your websites from the swap partition.

Here the interviewer takes some liberties, since this isn't a criminal investigation, to throw off the suspect. The hope is that the suspect will reveal what method was used to wipe the system and indeed that he was responsible at all.

Additional Information

The article by Dr. Anton Chuvakin regarding bmap and data hiding under Linux can be found at <http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html>.

Source and binaries for bmap were issued by Scyld Computing and had been posted at <ftp://ftp.scyld.com/pub/forensic_computing/bmap> but they have since been taken down. Source is posted at <http://www.madchat.org/crypto/stegano/unix/covert> however reliability cannot be guaranteed and caution should be used.

If interested, the official gzip specification can be found in RFC 1952 at <http://www.faqs.org/rfcs/rfc1952.html>.

Part 2
Analysis of an Unknown System

Synopsis of Case Facts

On 3/5/2003 the Crime Lab received the source media referenced below for forensic evaluation, per the request of Detective David Thome. Based on evidence recovered from a stolen motor vehicle, Detective Thome is conducting an investigation regarding 2nd Degree Forgery and requested that a search be done on the storage media for information relating to this investigation, i.e. finding evidence of manufactured checks and IDs, and of checks executed. As described by Detective Thome, the details of the case are as follows:

> [Mr. X] attempted to pass a manufactured and forged check at Cash 'N Go on January 4, 2003 in the amount of $460.70 on the account of [Company A] through Pinnacle Bank. [Ms. Y] later passed a check on that same date for the same amount, same account, same bank. X and Y were later arrested in Marshall, IL for possession of a stolen vehicle, forgery and other charges. A subsequent search of the vehicle produces forged instruments, false Nebraska IDs manufactured checks on various accounts with some executed and some practiced on. Included in these items seized was equipment used to manufacture checks and false IDs.

Equipment Used

Imaging-
Digital Intelligence F.R.E.D.
- Intel P-III 937MHz processor
- 512MB PC133 SDRAM
- 3COM 3C996B-T PCI Gigabit NIC
- Adaptec 29160 Ultra160 PCI SCSI Adapter
  - GSI Fastbloc® hardware write blocker
- Maxtor 34098H4 40GB Ultra ATA/100 hard drive

Storage-
Snap Server 4400 network attached storage appliance

Analysis-
Dell Precision 650 Workstation
- Dual Intel Xeon 2.8GHz processors
- 2GB PC2100 DDR RAM
- Integrated Intel PRO/1000 NIC
- Integrated LSI Logic U320 SCSI Controller
  - (1) Seagate ST336753LW hdd running Windows 2000
  - (1) Seagate ST336753LW hdd running Red Hat 9

Source Media

The evidence listed below is the computer and electronic media evidence submitted in this case.

Lab Case #: 2003C-0616	Agency Case #: P030074
Item #: 1	Town of Incident: Hastings
Description as Submitted:	Hewlett Packard Pavilion 503N computer S/N MX240B0371
Was seal found intact? ☒ Yes ☐ No	
Actual Submission Contents: ☒ same as above (if not, describe below)	
40GB Western Digital hdd S/N WMAAT8664598	

I first removed the evidence from the lab evidence storage area. The computer was stored in a cardboard box, which was sealed with a combination of evidence tape and clear tape. I opened the box and removed the enclosed computer. The box also included a keyboard, mouse, a USB cable attached to the computer and a power strip to which the computer, a (presumed) monitor power cable and an HP printer power adapter were all connected.

I removed all cables from the computer and removed the outer chassis case. Using a permanent marker, I labeled the inside chassis with my initials, the date and time, lab case number, lab submittal number and item number. This provides for two things, the first being accountability. If someone should open the computer's case in the future they will see that I have accessed it before them. If an issue should arise such as tampering or vandalism there is a record directly on the machine as to who was responsible for ensuring its integrity. The second thing this helps is future identification of the computer. Lawyers like to make use of "props" and hold evidence in front of a jury while they are talking about it. If asked "Is this the computer you processed on x/y/z date?" I have really no way to know, given the sheer volume of evidence I process in a year. Labeling and initialing the computer allows me to answer with great certainty, even years later, that a given computer was indeed handled and processed by me.

I then disconnected the power and data cables from the single hard drive and labeled each connector with my initials and the hard drive number (1). I removed the hard drive from the computer and labeled it with my initials, the date and time, lab case number, lab submittal number, evidence item number and hard drive number. This again allows for accountability and identification during future court proceedings.

I took the hard drive to my imaging machine and attached it using a Fastbloc hardware write blocking device by Guidance Software, Inc. The Fastbloc allows an examiner to attach an EIDE hard drive to a Windows system without fear of altering the evidence in any way. It does this by use of an internal bridge which accepts all system calls to the attached drive. The device then sends back the appropriate success message to the system, without actually passing any data to the hard drive. The only calls allowed to access the drive are read calls. In this way the Fastbloc acts as a data "flap," allowing information to move in only one direction.

I booted the imaging machine to Windows and loaded EnCase Forensic Edition v4. Using the standard Imaging Worksheet, form NSPCCU-3, I recorded the drive information (see below). I checked the manufacturer's website for drive specifications (Figure 12) and compared those against the values EnCase reported (Figure 13). The manufacturer's specs were attached to the Imaging Worksheet and placed in the file.

Physical Specifications	
Formatted Capacity [1]	40,020 MB
Interface	40-pin EIDE
Actuator Type	Rotary Voice Coil
Number of Disks	1
Data Surfaces	2
Number of Heads	2
Bytes Per Sector	512
User Sectors Per Drive	78,165,360
Servo Type	Embedded
Recording Method EPR4	Rate 16/17 PRML
ECC	Reed Solomon
Head Park [2]	Automatic

Figure 12: Manufacturer's drive specs

Total Size: 40,020,664,320 bytes (37.3GB)
Total Sectors: 78,165,360

Figure 13: EnCase drive information

IMAGING WORKSHEET

Lab Case #: 2003C-0616	Submittal #: 1	Item #: 1	Examiner: McGurk

Computer Information:

Make: Hewlett Packard	Model: Pavilion 500 #P9849A	Serial #: MX240B0371	Case Style: Tower
BIOS Make: Phoenix	BIOS Version: 4.06	BIOS Serial #: N/A	BIOS Password: Disabled
Actual Date: 06/30/03	CMOS Date: 06/30/03	Actual Time: 1634	CMOS Time: 1532
Boot Sequence: FDD/HDD/CD/NET			

Hard Drive Information:

Hdd #: 1	Make: Western Digital	Model:WD400EB	Serial #: WMAAT8664598
IDE Channel: Primary, Cable Select			
Mfr's C / H / S: N/A	Mfr's LBA: 78165360		Mfr's Capacity: 40.0 GB

Encase Information:

Physical				Logical			
Access: Fastbloc		Sectors: 78165360					
Size: 37.3 GB		C/H/S: N/A					
Code	Type	Sectors	Size	LP Label	System	Free	Size
0B	FAT32	10538640	5.0 GB	N/A	FAT32	1.2 GB	5.0 GB
7	NTFS	67601520	32.2 GB	HP_PAVILION	NTFS	20.8 GB	32.2 GB

Forensic Programs Used:

Name		DOS	WIN	**Name**	
Encase	v.	N/A	4.13	v.	

Acquire Information:

Dest. : Lab1	Number of Image Files Created: 12	Total Elapsed Time: 2:29:00

With drive specs verified, I began imaging the drive to NAS server "Lab1." Lab1 is a 780GB volume using a raid 0 array comprised of six 180GB EIDE drives. The volume is available via an SMB share which is mapped to drive X:\ on the forensic machine. I created a new folder on X:\ to hold this case, using the case number and suspects' names, and then created a subfolder to contain the image files.

I then created a new case in EnCase called "Acquire" and entered the requested information (Figure 14).

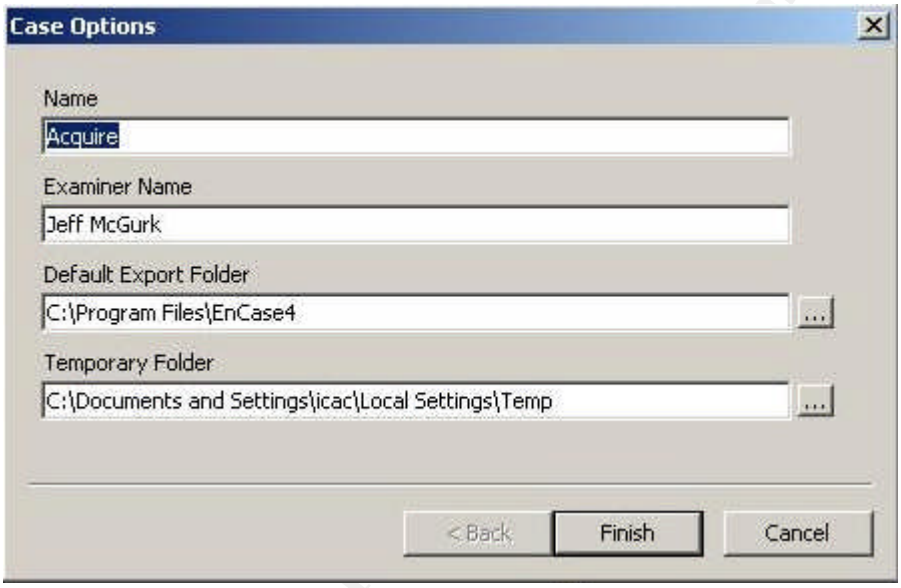

Figure 14: Create a new case in EnCase

The next step was to add the physical device to the newly created case before imaging it. Previous versions of EnCase allowed the examiner to simply specify a device to be imaged and proceed, however version 4 implements the extra steps of creating a new case and adding the device before imaging.
I clicked "Add Device" and in the resulting window chose "Local Drives" (Figure 15). Upon clicking "Next" the "Add Device" window closed and the "Choose Devices" window opened. I chose the evidence drive from the list and followed the prompts to finish.

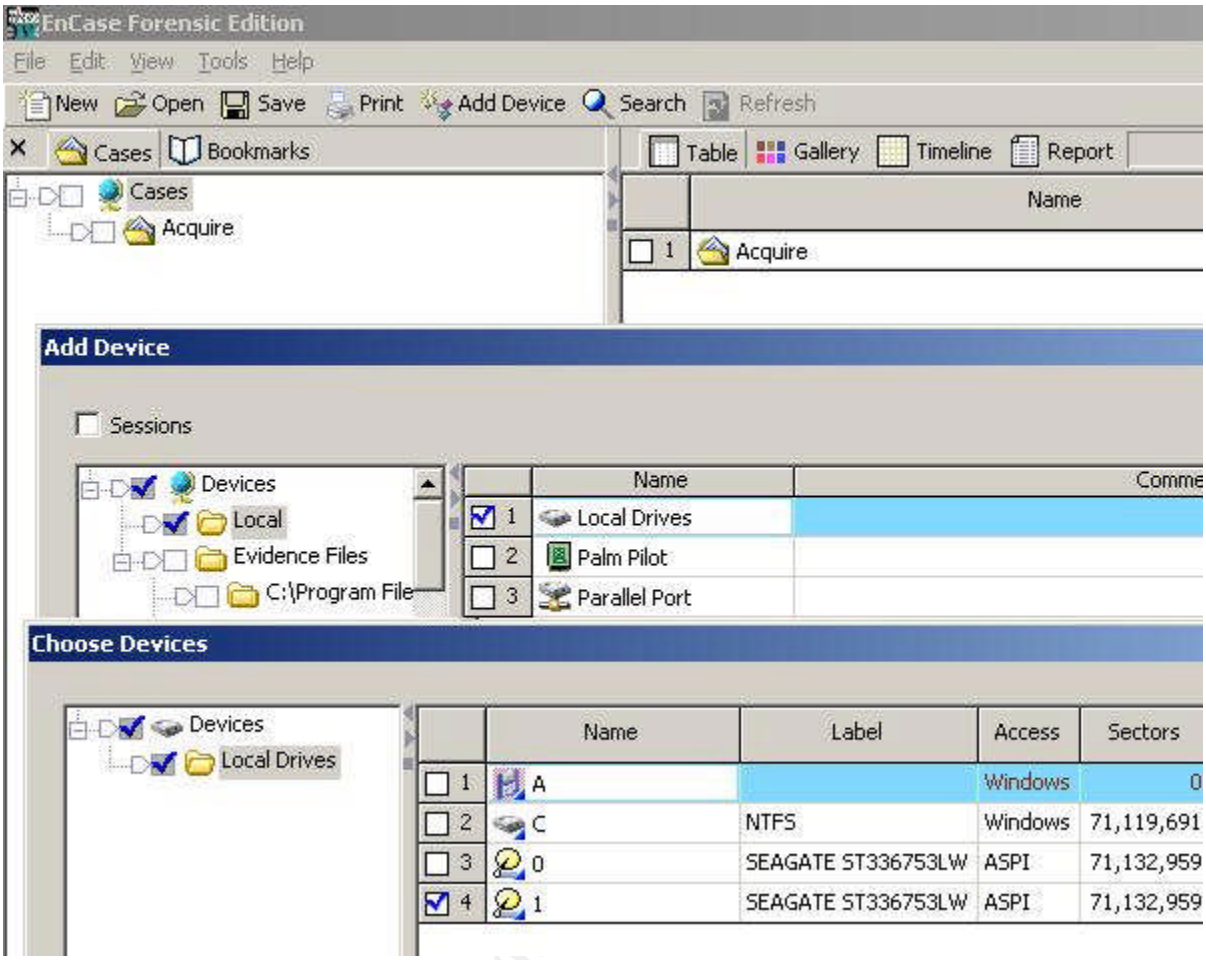

Figure 15: Add new device to case

With the evidence added to EnCase I was now ready to create my image file. Upon highlighting the evidence drive a new “Acquire” button appeared on the menu bar (Figure 16). I clicked “Acquire” and an “After Acquisition” window appeared.

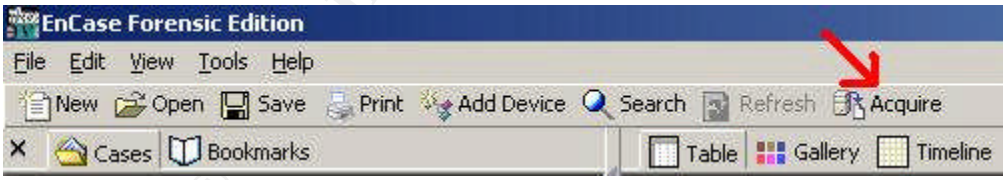

Figure 16: EnCase "Acquire" button

I chose “Replace source device” and clicked next, bringing up the “Options” window. I entered my case information (Figure 17) and clicked “Finish” to begin imaging.

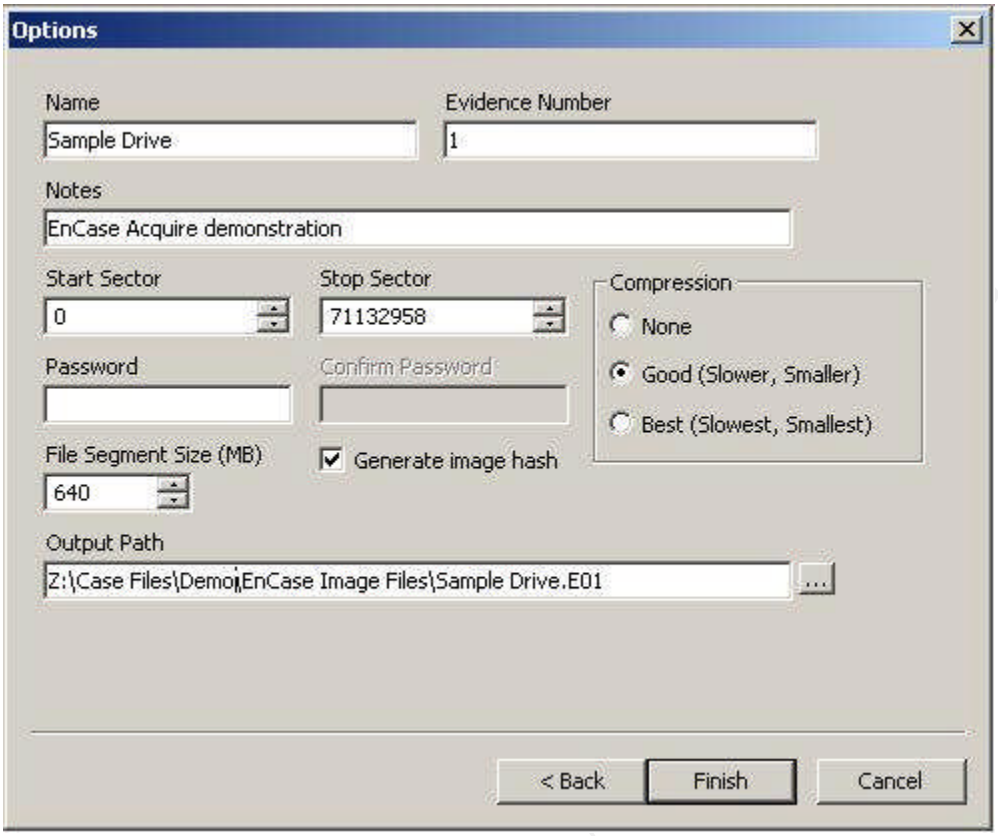

Figure 17: "Options" window for entering image description and settings

While the drive was imaging, I powered on the suspect machine to check the BIOS settings and recorded the settings on the Imaging Worksheet (above). The most commonly used information contained in the BIOS is the date and time of the system. This needs to be compared to the actual date and time to help determine the accuracy of the times stored on the hard drive. While matching date/time information does not guarantee accurate times it is advised to approach all times as suspect if the current system time does not match the current "real world" time. The BIOS time was found to be 62 minutes off. This is discussed below in the Timeline section.

The imaging finished successfully in 2 hours, 29 minutes. I then disconnected the drive from my machine and returned it to the suspects' computer. The computer was then resealed with tape covering the drive bays, chassis access points and power connector. I returned the computer to the cardboard box and sealed the box with tape. I then returned the box to the lab evidence storage area.

Analysis

The previously created image files sat on the NAS server until 01/07/2004, when this case came up in the rotation. I loaded the image files into EnCase v4.16a and allowed EnCase to verify the file integrity. EnCase does this by calculating a new MD5 hash on

the present data and checking it against a value calculated at the time of imaging (Figure 18). Since these hash values match it is computationally infeasible that even one bit has changed since the day the imaging took place.

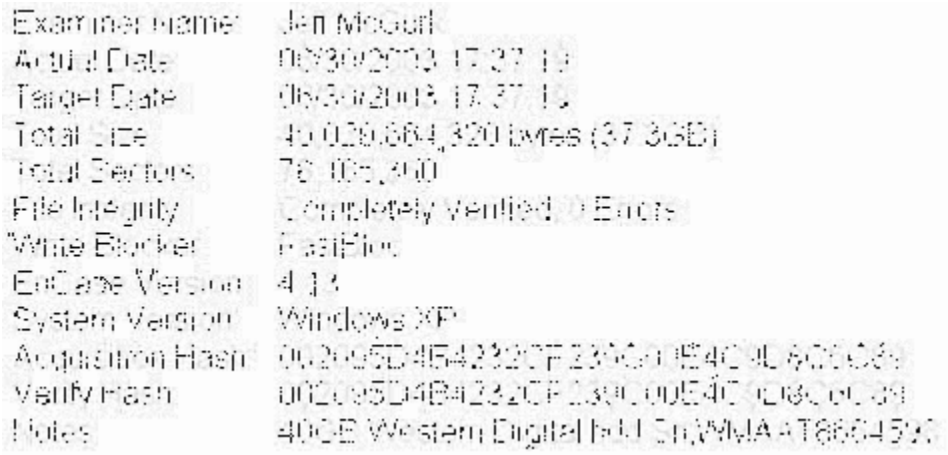

Figure 18: EnCase information with verified hash value

After the evidence had verified as good, I checked the “system” and “software” registry hives for system information. There were no network shares listed in the “Services/lanmanserver/Shares” registry key. Also, curiously, the “Registered User” and “Registered Organization” are blank. I began running the EnCase function for checking file signatures and calculating hash values on individual files. The signature checker compares each file's extension to a list of known extensions, and verifies that the file header matches for each given extension. The hash function calculates an MD5 hash value for each file on the drive. This allows comparison against a list of known files such as Windows .cab files, .dll files, etc and any such known files can be ignored as irrelevant to the case at hand. This is because every install of Windows XP is going to have a certain number of files in common with every other installation of the same version. These files will have the same hash value on every single machine, and any file that matches that hash value can be said with mathematical certainty to contain exactly the same data. The National Institute of Standards and Technology maintains databases of such files at <http:// www.nsrl.nist.gov> for the purpose of separating these known files from the unknown.

After the hashing and signature checking was complete, I restored the drive to a clean lab drive and scanned for viruses using McAfee VirusScan with DAT 4312. No viruses or trojans were found on the drive. I also manually checked the \SOFTWARE\Microsoft\Windows\CurrentVersion\Run registry key for programs loaded at system startup. I performed an internet search for each program and my results are listed in Appendix A.

All programs listed to run at startup are commonly found applications, not known to contain malware. McAfee also found no signs of malware on the system's hard drive. It is therefore believed that the suspect system has not been compromised and should be checked only for evidence of the suspected forgery offenses.

I next ran an EnCase EnScript that I prepared to search for printer spool files. "EnScript is a powerful macro-programming language and API developed by Shawn McCreight that is designed to work within the EnCase® environment. Hundreds of existing EnScript programs perform such tasks as finding and extracting all image files, instant message chat logs, or spreadsheet files within unallocated clusters."[4] This particular script uses EMF-type headers known to be used in Microsoft Windows printer spooling to check for potentially printed information stored on the hard drive. The information is then checked manually to see if any viewable data exists. The current EnCase "Graphics File Finder" EnScript covers these printer spool files, but at the time of my exam this was not the case and I needed to craft my own (see appendix B.) No data was able to be viewed using this technique. This is common because printer spools are temporary files, which are deleted and overwritten quite quickly.

My next step was to check for images visible in the EnCase gallery. The EnCase gallery provides a quick access method to view all of the recognized graphic files in allocated space (Figure 19). I browsed the gallery view and bookmarked several files relating to false IDs.

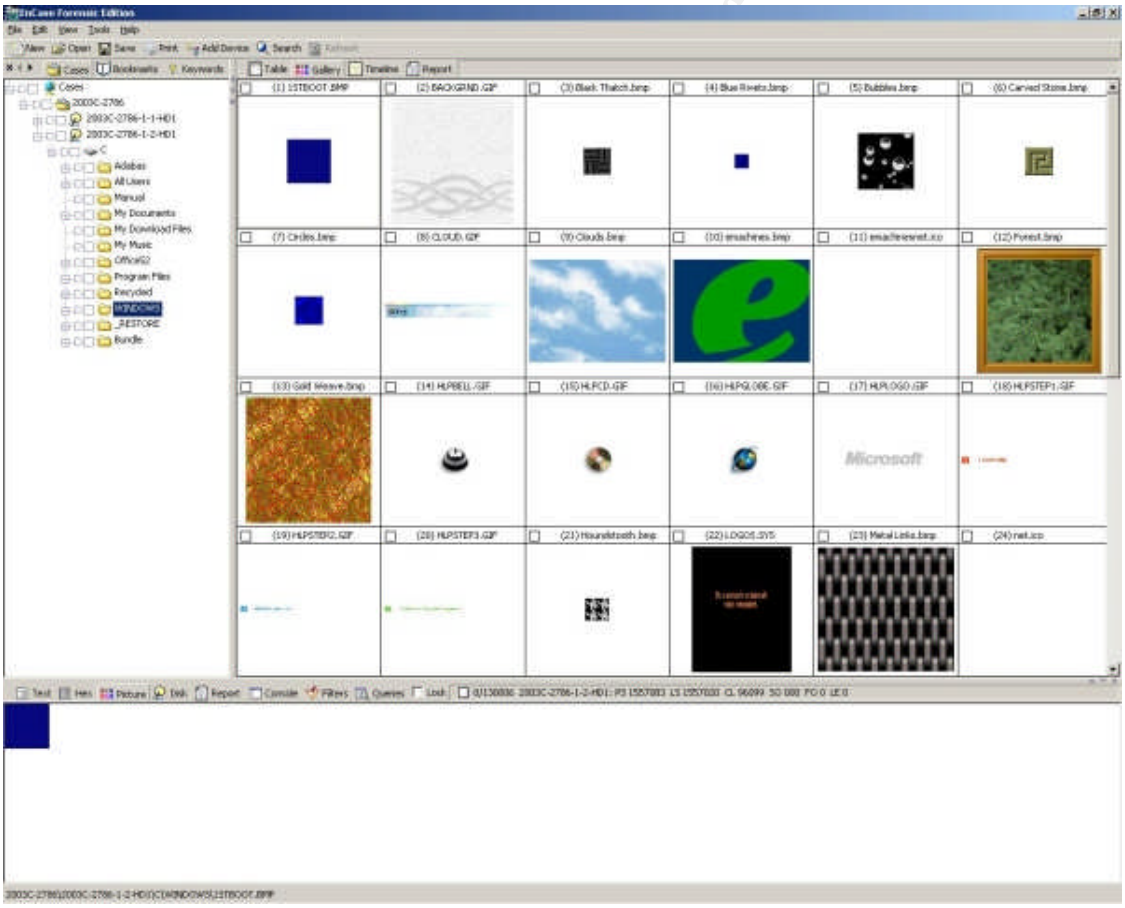

Figure 19: EnCase gallery view

[4] Guidance Software. "EnCase EnScript - designed by Shawn McCreight". <http://www.guidancesoftware.com/support/enscript/index.shtm> (5 April, 2004)

I next searched for evidence of the known checks by string searching for "460.70," "[Company A]" and "Pinnacle." No hits were found for "460.70." "[Company A]" resulted in one hit from unallocated space. I bookmarked this result for inclusion in my report to Detective Thome. This information has been purposely excluded from this paper, but it consisted of the account name, [Company A], the company address, the bank name, Pinnacle, bank address and account number.

The "Pinnacle" results included some data believed to be related to printing activity, located in streams containing headers of "PDL." I bookmarked the readable text and tried to view the streams as they might have been printed on the page. Most printers require that a page to be printed be passed as an image instead of text, as if someone had taken a picture of the page and printed that instead. These images are often formatted as EMF graphics and can usually be recovered and viewed in any common image viewer. I checked for viewable images in the "PDL" hits and also searched for any EMF headers that may contain additional printed information. No print images were viewable using the above methods.

Unable to view the data contained in the "PDL" streams I posted a message to the Computer Forensics Investigators Digest (CFID) listserv asking if anyone had dealt with such data before. Rich Mason responded that he has not seen "PDL", but that ".%-12345X" is a common RAW print header, and that he has also seen "@PJL" used quite frequently as well. I searched for both headers, which resulted in about 600 hits. Reviewing these hits resulted in no viewable print data, but several check transactions are clearly visible within the text. I bookmarked these hits for delivery to the investigator.

To attempt to find information as the suspect saw it, I booted a restored copy of the drive in VMware.

> VMware Workstation works by enabling multiple operating systems and their applications to run concurrently on a single physical machine. These operating systems and applications are isolated in secure virtual machines that co-exist on a single piece of hardware. The VMware virtualization layer maps the physical hardware resources to the virtual machine's resources, so each virtual machine has its own CPU, memory, disks, I/O devices, etc. Virtual machines are the full equivalent of a standard x86 machine.[5]

By cloning the suspect hard drive to a clean lab drive I was able to bring up the suspects' operating system inside the virtual machine just as if the drive were physically attached to another computer. These suspects ran Windows XP Home Edition and the configuration of the virtual machine required that Windows activation be rerun. Windows activation is explained at <http://www.microsoft.com/windowsxp/pro/evaluation/overviews/activation.asp> This new copy protection scheme that Microsoft introduced with the XP line (Windows and Office) works by way of activation codes. The installation routine generates a supposedly unique Installation ID number from a combination of the Product ID and various hardware components of the machine it is installed on. The user then either

[5] Vmware Inc. "Products -- VMware Workstation 4.5". <http://www.vmware.com/products/desktop/ws_features.html> (6 April, 2004)

places a telephone call or accesses an internet site and provides Microsoft with the Installation ID and they in turn send back an activation code to unlock the software. This way, only one machine at a time can use a particular Product ID, instead of the old days when any number of people could share one Windows CD and ID number. For my purposes it prevented me from booting Windows inside the virtual machine because the hardware configuration did not match that of the original installation and thus Windows could tell that the Installation code was not valid for my setup. Windows XP Service pack 1 introduced a 3 day grace period for reactivating, but the suspect's machine was not running SP1.

I instead booted into "Safe Mode" to avoid performing a reactivation. In a program called PhotoImpression, I discovered the existence of several files with the extension "PSF" that contain albums of photos. Several of these files, saved in the "Owner" user directory, contain albums of images that could be used to create false Nebraska IDs (Figure 20). Within EnCase, I ran a graphic recovery EnScript (Appendix C) on all PSF files to pull out any JPG format images within them. This resulted in 2223 hits which I went through, bookmarking all ID related images.

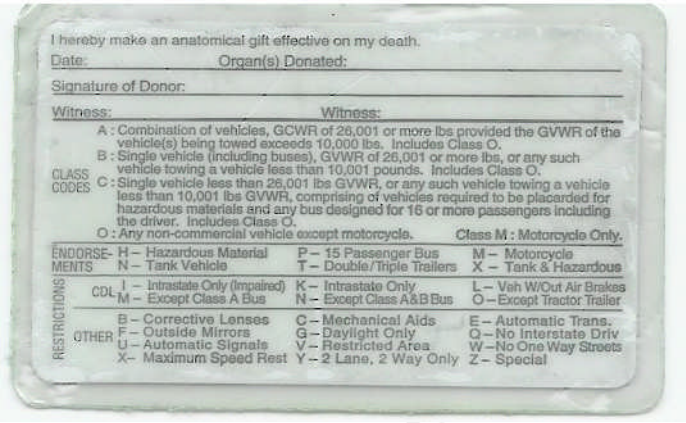

Figure 20: Example of fake ID images

The system also contained check writing programs "VersaCheck" and "My Checkwriter." I located a VersaCheck data file called DSETUP.vdf and installed VersaCheck on my lab analysis machine to view the file and print any recoverable information. The file appeared to contain account data for printing checks in the name of [Mr. Z] and [Company B]. Two accounts are set up for [Mr. Z], one for Pinnacle Bank-Gretna at 6145 Havelock Ave in Lincoln, and one for Pinnacle Bank-Gretna at 70th & Adams in Lincoln. The [Company B] account is drawn from Pinnacle Bank-Gretna at 6145 Havelock Ave in Lincoln. I printed all checks listed in the registers of all three accounts and attached them to the report provided to Detective Thome.

After printing the checks, I reviewed the unallocated space on my lab machine to look for what print information was sent by the VersaCheck software. The results showed that "NAME="VersaCheck" appeared at the beginning of each print job that I submitted. I searched the unallocated space and pagefile.sys of the suspect drive for the same string, but no hits were found. I also tried additional strings of "JOB NAME" and

“JOBNAME”. All results for these searches occurred in line with my previous “@PJL” search, so I canceled the search after 1 hour with 1hr 10min remaining.

I next searched the unallocated space and pagefile for the string “AND ##/100” to find any checks that may have been missed by the “Pinnacle” search. Several partial check transactions were located and added to the bookmarks. These appear to be in the same format as those located in the Versacheck .vdf files.

Using Encase's “copy/unerase” command, I exported all files from “Program Files\MySoftware\MyCheckBook\Data.” I then used a hex editor to change the status of checks within .mcb files to be reprinted. In loading the files into the MyCheckwriter software, I noticed a simple yes or no flag indicating if a check is “selected for Printing.” By using the hex editor, I saw that each record had a “0” at offset 702. Thinking that this may indicate the record was not selected for printing, I changed each “0” to a “1.” In “00[Name-Withheld1].mcb,” this meant printing 1 check, #3346 to [Ms. Y]. The register did not indicate that this check had previously been printed. A prompt appeared asking to choose the owner of checks, with choices of [Mr. Z] or “Sample Company.” “Sample Company” has no associated document name to print from, so I chose [Mr. Z], which has one document called “Test.” The check printed successfully and was attached to the report.

“00[Name-Withheld2].mcb” contains no transactions in the register.

In “00[Name-Withheld3].mcb,” I chose to print 1 check, #100 made payable to “Me.” The register indicates that the check was printed on 12/24/2002.

None of the checks found include the known [Company A] check described by Detective Thome. To attempt to locate additional information about the [Company A] account, I exported an extra copy of DSETUP.vdf, then overwrote from hex “040108077E” with data from the [Company A] search hit, saving the resulting file as [Company A].vdf. I loaded [Company A].vdf into VersaCheck and screen captured several screens (Figure 21). On the “Write Check” screen, the “Pay to the Order of” field uses a drop-down menu that included the name of suspect [Mr. X]. When I chose this name the remaining fields automatically filled in, presumably with the data used for the last check made out to this payee.

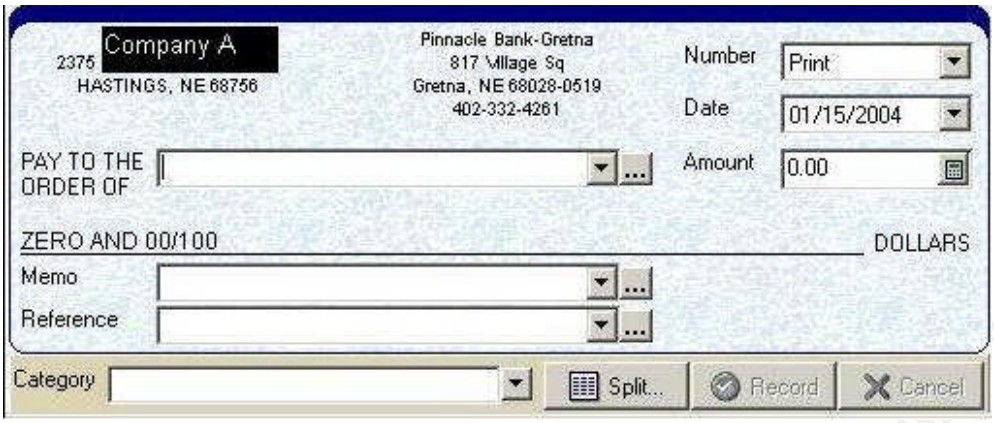

Figure 21: [Company A].vdf screen shot

I had hoped to provide a sample to Detective Thome to compare with the 2 checks that were known to have been passed on 1/04/2003. Unfortunately, since the file was constructed using portions of two unrelated files, some essential information was missing and I was unable to print any sample checks. However, the account number is contained in the bookmarked text of the [Company A] search hit and this should prove helpful, if it matches the account numbers found on the aforementioned checks.

I concluded my analysis by verifying that all account information located during the various string searches, including account numbers, had been bookmarked appropriately. I then exported my bookmarks to an html page for delivery on CD-ROM. Included among these bookmarks was subscriber information for a Charter Communications broadband Internet account that was located during a search of the Windows registry. A search of the Yahoo! White Pages verified that the name, address and phone number are valid. The name also matches an ID found in the ".PSF" albums mentioned above, though the address is different. It is unclear if the computer was stolen from this individual, along with his wallet (or at least his driver's license), or if this is another accomplice.

String Search

As mentioned above several string searches were performed to locate data in various places on the drive. EnCase allows the examiner to input any number of keywords to search across the media (Figure 22). The keywords are saved so that common terms can be reused from case to case.

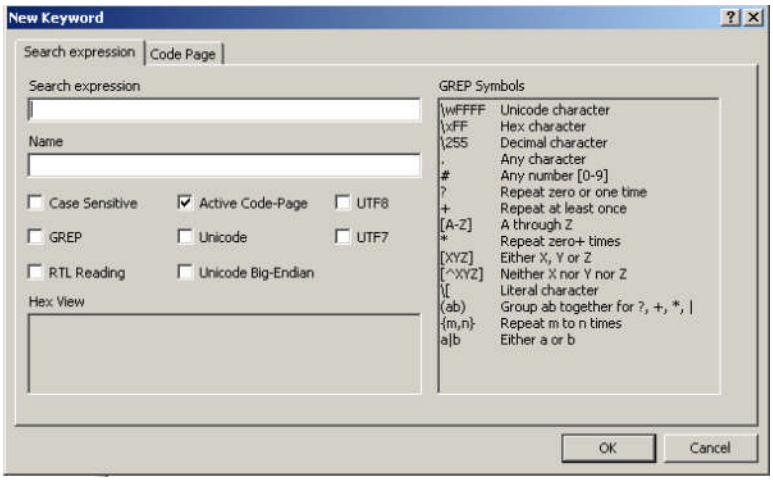

Figure 22: EnCase keyword entry

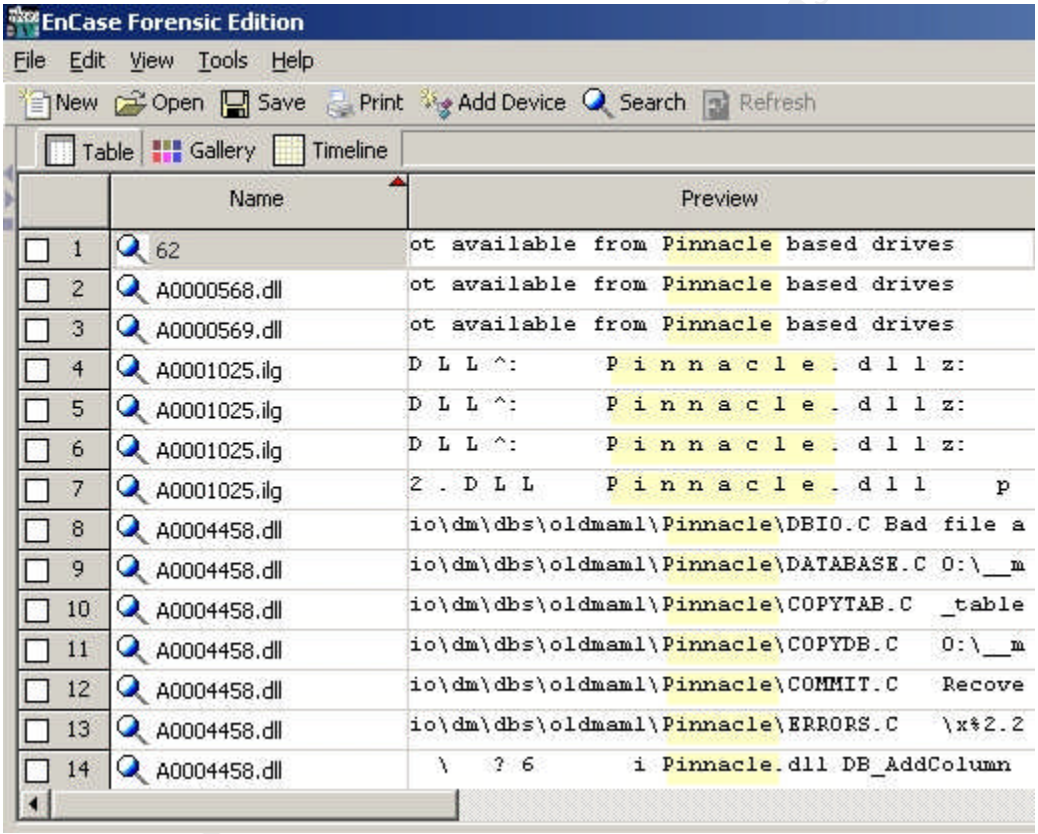

Figure 23: Search results for keyword "Pinnacle"

I first input 3 keywords, “460.70,” “[Company A]” and “Pinnacle,” that were reported to be included on forged checks passed by the suspects. EnCase then went sector by sector over the physical drive, as well as logically through each file, in search of my keywords. The results were presented in a table showing the context in which each hit occurred (Figure 23). I reviewed the hits for all 3 keywords, finding one hit for “[Company A]” in unallocated space and numerous hits for the word “Pinnacle.” I

bookmarked the relevant results for inclusion in my report to Detective Thome. EnCase bookmarks, as the name suggests, provide a means to jump instantly to an important piece of information. Bookmarks may then be arranged into a final report of all of an examiner's findings.

I next wanted to search for recoverable print data and added the keywords ".%-12345X" and "@PJL" that were suggested by Rich Mason on CFID. I reviewed the 600 hits for the 2 print related keywords, but could not view any of the print images discussed above. I instead bookmarked the text of the hits that appeared to contain check information. This information included bank names, account numbers, "Pay to the order of", etc. As indicated in the Analysis section above, I also searched for print data using keywords "NAME="VersaCheck," "JOB NAME" and "JOBNAME." These produced no additional information.

Since data in unallocated space gets overwritten gradually I decided to search for any partial check data that may have been missed by previous searches. I used a keyword of "and ##/100" to locate the amount field of the checks. The # symbol is a wildcard that matches any single digit, so the keyword becomes "and " followed by any two digits followed by "/100". This resulted in several partial check entries that I then bookmarked.

Timeline

A timeline analysis was not pertinent to the investigation, but has been included here for certification purposes. The BIOS check during the imaging phase revealed a difference of 62 minutes between the system clock and the actual time. 2 minutes is acceptable deviation, but the additional 60 minutes requires explanation. This is almost certainly due to the fact that the computer was seized during the months of Central Standard Time, but imaging took place during Central Daylight Time. Since the system was in police custody on April 27, 2003 the clock was not "sprung ahead" and still reflects CST, or -6 hours from GMT. The actual time observed during imaging reflects the -5 hours for Central Daylight Time. It may be assumed, barring other indicators, that dates and times were accurately recorded to the hard drive, and EnCase v4 translates them according to the appropriate time zone setting.

A brief chronology of the system which will be detailed below is:
Late July 2002 – Hewlett Packard began OEM setup of the computer system. Most of the patches present on the system were applied at this time.
08/03/2002 – Setup completed and Windows metafiles were created.
10/24/2002 – Probably close to purchase date of the computer. Windows records this as the official install date, and pagefile.sys and hiberfil.sys indicate Windows was probably first booted on this date.
12/22/2002 – Earliest files related to fake IDs were created. Two MyCheckBook data files created.
12/24/2002 – Third MyCheckBook data file created
12/27/2002 – Versacheck software installed
01/09/2003 – Last ID file accessed
01/10/2003 – Last logoff and last shutdown.

There was some initial confusion concerning the install date of the system, but closer study seemed to explain everything. I had first checked the registry to find the install date in the SOFTWARE\Microsoft\Windows NT \CurrentVersion\InstallDate key. As shown in Figure 24, the date stored there was 10/24/2002.

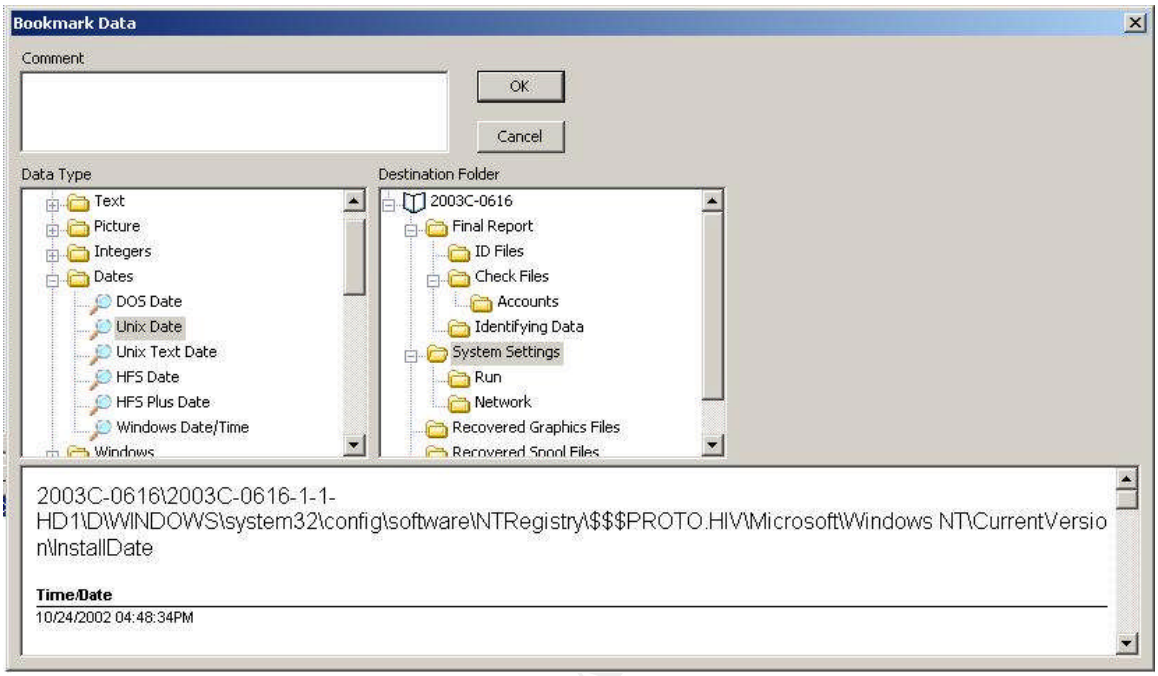

Figure 24: Install Date as stored in Windows Registry

The confusion began when I checked the date/time stamps on the system files such as $Boot, $MFT, etc. These files are created when an NTFS volume is first formatted, and for the average user this would normally match the date that Windows was installed. However, as seen in Figure 25, these files were all created 08/03/2002 at 9:22 PM.

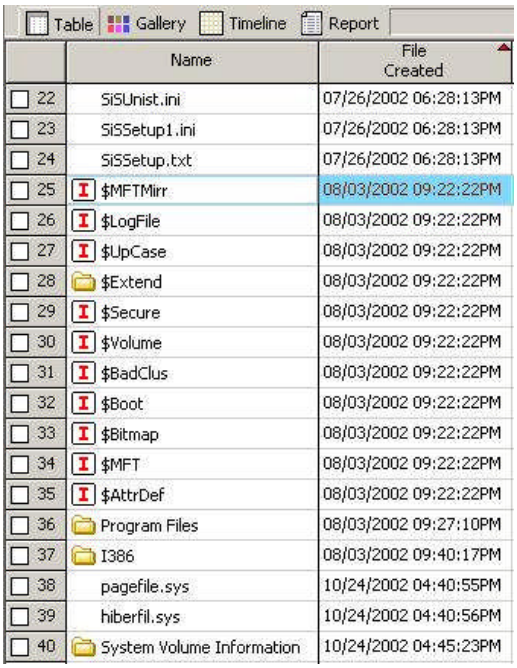

Table | Gallery | Timeline | Report

	Name	File Created
22	SiSUnist.ini	07/26/2002 06:28:13PM
23	SiSSetup1.ini	07/26/2002 06:28:13PM
24	SiSSetup.txt	07/26/2002 06:28:13PM
25	$MFTMirr	08/03/2002 09:22:22PM
26	$LogFile	08/03/2002 09:22:22PM
27	$UpCase	08/03/2002 09:22:22PM
28	$Extend	08/03/2002 09:22:22PM
29	$Secure	08/03/2002 09:22:22PM
30	$Volume	08/03/2002 09:22:22PM
31	$BadClus	08/03/2002 09:22:22PM
32	$Boot	08/03/2002 09:22:22PM
33	$Bitmap	08/03/2002 09:22:22PM
34	$MFT	08/03/2002 09:22:22PM
35	$AttrDef	08/03/2002 09:22:22PM
36	Program Files	08/03/2002 09:27:10PM
37	I386	08/03/2002 09:40:17PM
38	pagefile.sys	10/24/2002 04:40:55PM
39	hiberfil.sys	10/24/2002 04:40:56PM
40	System Volume Information	10/24/2002 04:45:23PM

Figure 25: Created dates of Windows metafiles

It can also be seen in Figure 25 that pagefile.sys and hiberfil.sys were both created on 10/24/2002. Pagefile.sys is a swap file used for swapping memory data and hiberfil.sys is used to store the contents of RAM when Windows hibernates. These files are normally first created when Windows is booted for the first time, although they can be recreated later if deleted by the user. I did some more exploration to try to come up with an explanation and found a file named software.log. Software.log is, as it sounds, a log file of the running of various programs. The first few lines are:

```
2002-07-23 18:33:05 HP_Build_PrepBlock_WW_XP_0000-60.exe
2002-07-24 08:58:43 beginbuild_WW_XP_0000-01.exe
2002-07-24 08:59:05 FullScreen_WW_XP_0000-10.exe
2002-07-24 08:59:26 HPSysInfo_WW_XP_00000741_D-04.exe
2002-07-24 08:59:49 Fall02_all_boards_EN_XP_00000000_A-01.exe
```

This log continues until the 26th when it ends with "2002-07-26 13:38:05 HP_EndBuild_WW_XP_0000-36.exe." This suggests to me that the computer was undergoing some configuration as might be done by HP prior to deploying the system.

I therefore feel that the precise dates and times during this early stage are unreliable. Original equipment manufacturers such as Hewlett Packard often use special OEM CD's with Windows pre-configuration routines. It is enough to know that around the end of July 2002 the system was undergoing this sort of pre-configuration, either at the factory or by a system restore CD used by the owner. 10/24/2002 is quite reliably the date on which configuration was complete and Windows was booted into interactive GUI mode.

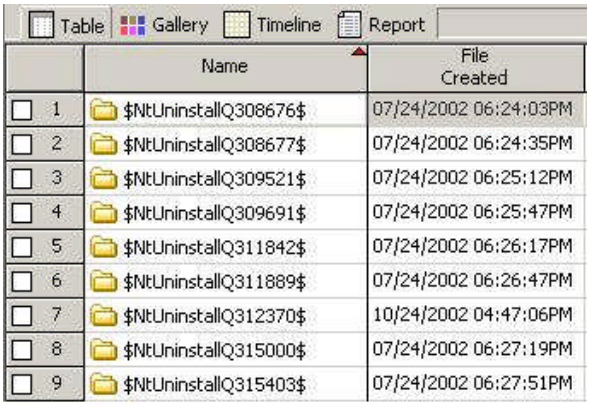

Table | Gallery | Timeline | Report

	Name	File Created
1	$NtUninstallQ308676$	07/24/2002 06:24:03PM
2	$NtUninstallQ308677$	07/24/2002 06:24:35PM
3	$NtUninstallQ309521$	07/24/2002 06:25:12PM
4	$NtUninstallQ309691$	07/24/2002 06:25:47PM
5	$NtUninstallQ311842$	07/24/2002 06:26:17PM
6	$NtUninstallQ311889$	07/24/2002 06:26:47PM
7	$NtUninstallQ312370$	10/24/2002 04:47:06PM
8	$NtUninstallQ315000$	07/24/2002 06:27:19PM
9	$NtUninstallQ315403$	07/24/2002 06:27:51PM

Figure 26: Dates of Windows Updates

As seen in Figure 26, most updates were applied during the configuration period on 07/24/2002. One update, Q312370, was applied on 10/24/2002 during the final install of the operating system.

The next activity of interest chronologically is when any apparent forgery may have begun. The files that appeared to be used for manufacturing fake IDs first appeared on the drive on 12/22/02 (Figure 27).

Last Accessed	Last Written	File Created	Full Path
01/07/03 06:54 am	12/22/02 05:40 am	12/22/02 05:40 am	\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Dec 2002\scan0001.jpg
12/30/02 10:04 am	12/22/02 01:52 pm	12/22/02 12:39 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\Picture 001.jpg
12/30/02 10:04 am	12/24/02 02:43 am	12/22/02 12:59 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\dm.PSF
12/30/02 10:04 am	12/22/02 02:32 pm	12/22/02 02:10 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\dam.PSF
12/30/02 10:04 am	12/22/02 02:34 pm	12/22/02 02:33 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\Dec.PSF
12/30/02 10:04 am	12/22/02 02:35 pm	12/22/02 02:35 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\Picture 002.jpg
12/30/02 08:36 am	12/22/02 04:56 pm	12/22/02 04:19 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\Dec 2002\scan0002.jpg
12/30/02 08:36 am	12/22/02 04:35 pm	12/22/02 04:33 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\Dec 2002\scan0003.jpg
12/30/02 10:04 am	12/23/02 10:14 am	12/22/02 10:35 pm	\Documents and Settings\Owner\My Documents\My Pictures\Picture\undo001.jpg
12/27/02 08:01 pm	12/22/02 02:40 pm	12/22/02 11:00 pm	\Documents and Settings\Owner\Local Settings\Temp\~DEST\Dec22_08.JPG

Figure 27: Earliest ID files located

Activity on this date seems to be a lot of scanning and then manipulating various legitimate licenses. File scan0001.jpg has the earliest created date, with scan0002.jpg and scan0003.jpg coming sequentially down the line. Several files had last written times later than their creation times, suggesting that the images had been edited in some fashion. The last file in the above list, Dec22_08.JPG, illustrates a classic date/time anomaly. Notice that the created time was 2300 hours on 12/22, and the last written time was 1440 hours on the same day. The file was edited nearly 8 ½ hours BEFORE it was created!

This suggests that this is not the original copy of this file. The Windows created date reflects the date and time that the file was first created on this particular volume, while the written date reflects when the contents of the file were last edited, even if it was on a different volume than where the file currently resides. If the file was edited at 1440 and then moved to another disk at 2300, the result would be as shown in the above list.

The last ID related file to be created, written or accessed was Jan09#89.JPG on 01/09/03. This file contains the image of a signature. It seems logical that the suspects would create their IDs exactly how they wanted them and then add the signature to create the final product.

There were three data files for the MyCheckBook software located, two created on 12/22/02 and one created on 12/24/02. The MyCheckBook software itself had been removed, leaving only these data files behind. There were no registry keys or log files to indicate when the software may have been installed. The 3 remaining data files all have their individual last written times equal to their last accessed times, all between 1656 and 1824 on 12/24/02. It was probably not long after this that the software was removed.

Additionally, the only two allocated data files to contain VersaCheck information were Vcheck.vdf and DSETUP.vdf, both of which were created on 12/27/2002. This corresponds to a located registry key which states that VersaCheck was installed on the same date, 3 days after the last MyCheckBook file was accessed. Perhaps MyCheckBook did not meet the suspects' needs so they decided to try VersaCheck.

The event that is naturally expected to occur last on any computer system which is in the "off" state is the time at which it entered such a state. That is to say, when was this computer shut down or when was the plug pulled? If Windows is allowed to shut down properly the time at which this happens is recorded in a couple of ways. As seen in Figure 28 the last recorded logout for this system was recorded in the security event log at 1/10/03 at 1026. This matches the SYSTEM\ControlSet001\Control\Windows\ShutDownTime registry key which recorded 1/10/03 1026 as the last shutdown time of the system. The suspects were picked up shortly after this, with the computer in their possession, so it is probably an accurate reflection of the last time the system was used.

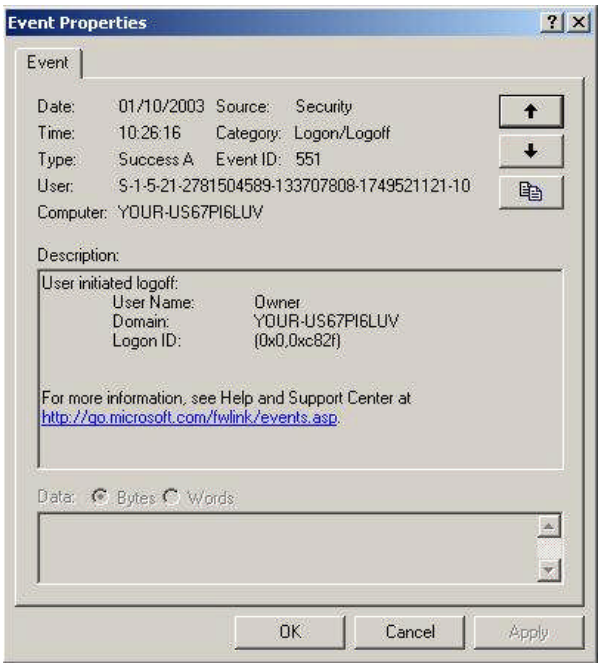

Figure 28: Last logout recorded by Security Event log

The full timeline of all 76000 files created a spreadsheet of over 15MB and is therefore not included in this report.

Deleted Files

One of the great things about the EnCase software is that it automatically attempts to recover as many deleted files as possible. The term "unallocated space" which I have used several times refers to all clusters that are not currently being taken up by active files (those that show up in Windows Explorer or DOS's *dir* list.) An unallocated cluster must satisfy one of two conditions: either it has never contained data or else data it once contained has been "deleted." I place deleted in quotes to emphasize the very principal of data forensics that data is not truly gone until it is overwritten with other data.

Files that have been deleted from an NTFS volume such as the suspects had can themselves satisfy one of multiple conditions, and EnCase handles them according to which one they satisfy.

Firstly, a file[6] may be deleted from its parent directory, in which case the file's entry in the parent directory is deleted, the entry in the Windows master file table ($MFT)[7] is marked as reusable and the clusters are marked available in the $Bitmap file. If

[6] Everything in NTFS, including directories, is treated as a file. Therefore the concepts described for files apply equally to directories.
[7] See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/base/master_file_table.asp for an explaination of the $MFT file.

EnCase finds an $MFT entry that has been deleted in this way it checks for the parent entry (every $MFT record indicates the $MFT record number of its parent directory) to see if it is still intact. If an intact record belonging to a directory is located then EnCase places the file back into this directory and indicates that the file has been deleted (Figure 29). One interesting point to note is that EnCase has no way of verifying if the directory residing in the $MFT record listed as the parent is in fact still the correct parent. If the actual parent were to be deleted and then a new directory given its spot in the $MFT then EnCase, and any other file recovery method, would be forced to assume that the new directory is the parent of the file in question. See Appendix D for more information.

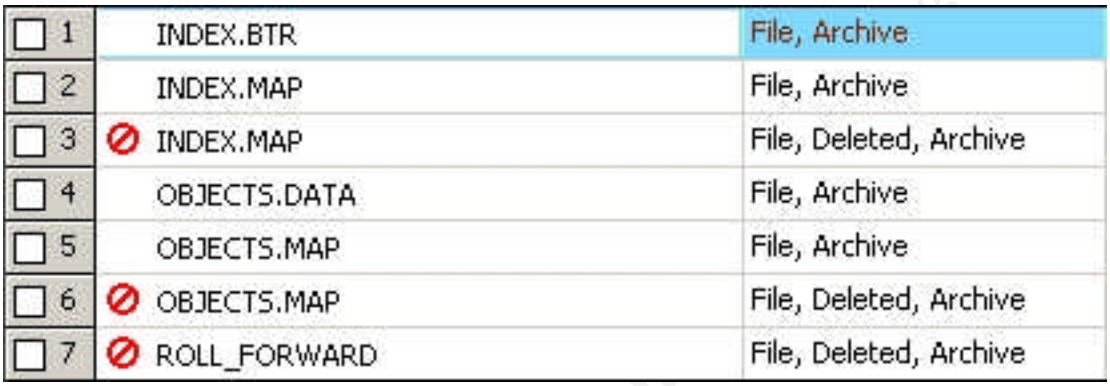

Figure 29: EnCase file list showing deleted files

A second possibility is that EnCase will encounter an $MFT record for a deleted file and check the parent entry, only to find it no longer in tact. In this case the file will be designated as lost. All files that fit this description are placed into a “Lost Files” folder in the EnCase file structure (Figure 30).

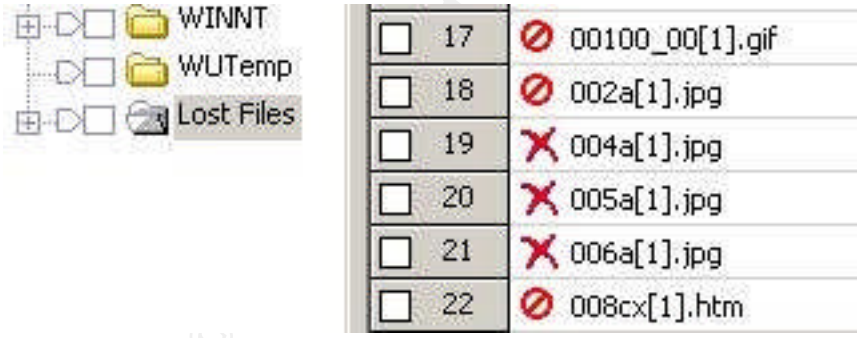

Figure 30: Example of EnCase "Lost Files"

The two above possibilities involve MFT records marked for reuse but which have not yet been overwritten. Additionally, the deleted file’s data remains in unallocated space until such time as it too is overwritten. Two more possibilities are if the MFT record has not yet been overwritten but the data in unallocated space has, or conversely if the MFT record is overwritten, but the data remains in unallocated space. EnCase designates the former files as “Deleted, Overwritten” and provides the original directory location of the deleted file as well as the location and name of the file that is now using the space on the disk. Figure 31 illustrates this in EnCase. The red X indicates that the

Shockwave Flash file has been overwritten, and the path along the bottom tells that pagefile.sys is the file that overwrote it.

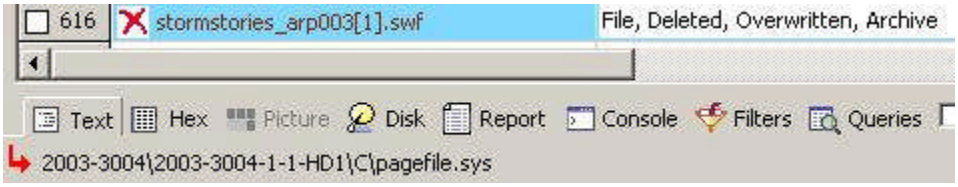

Figure 31: EnCase designation of deleted and overwritten files

The latter type of file, with MFT record gone but data remaining in unallocated space, describes all of the relevant deleted information located during my analysis. With the MFT record gone I was unable to recover filenames or dates/times associated with any of the data. This type of metadata is stored directly in the MFT record and once it is overwritten becomes extremely difficult to recover. The only hope is to locate any possible help outside the MFT such as .lnk files or transactions logged in $Logfile.

There was one file of interest which had been sent to the Recycler and had not been emptied yet. The file was called MyCheckBook20021222.bak and contained a PKzip structure with what appeared to be backups of one of the .mcb MyCheckWriter files. MyCheckBook20021222.bak was created 12/22/02 1527 and deleted 12/30/02 0800.

Conclusion

As a result of my forensic analysis of the above listed media, it appears that the submitted system was in fact used by [Mr. X], [Ms. Y] or both to commit the offense of forgery in the second degree, in violation of Nebraska Statute Section 28-603. Subsection 1 of section 28-603 states that

> Whoever, with intent to deceive or harm, falsely makes, completes, endorses, alters, or utters any written instrument which is or purports to be, or which is calculated to become or to represent if completed, a written instrument which does or may evidence, create, transfer, terminate, or otherwise affect a legal right, interest, obligation, or status, commits forgery in the second degree.[8]

An abundance of data for creating fake IDs and printing counterfeit checks was located all over the hard drive. The account numbers, names of account holders, and some sample checks were included in the report turned over to the investigator of the case. The dollar amount of the checks recovered off of the hard drive indicates that the suspects, if convicted, would be sentenced under subsection 2 of the above named section, which states that

> Forgery in the second degree is a Class III felony when the face value, or purported face value, or the amount of any proceeds wrongfully procured or intended to be procured by

8 Section 28-603, Nebraska State Statutes. 31 Aug., 2003. <http://statutes.unicam.state.ne.us/Corpus/Statutes/chap28/R2806003.html>(24 Feb., 2004)

the use of such instrument, is one thousand dollars or more.[9]

The checks made out to the names of the suspects that were recoverable and printable totaled $3966.09. In addition there was $11133.64 worth of checks made out to 2 other names, one of which was found among the images of fake IDs. These are most likely pseudonyms used by the suspects when cashing phony checks.

Additionally, an attempt should be made to ascertain the proper owner of the computer as there is a possibility that either it was stolen or else there may be an additional accomplice besides the two found in possession of the machine.

9 Ibid

Part 3
Legal Issues of Incident Handling

1. It is believed that John Price was producing and/or distributing unauthorized copies of copyrighted works in the form of audio and perhaps video recordings in violation of United States copyright law, Title 17, Section 106. Section 106 states that owners of copyrights hold the exclusive right to reproduce any work covered under such copyright. Section 503 of Title 17 provides that an infringer of copyright is civilly liable to the copyright holder for actual damages plus all profit realized by the infringing act. If proven to be a willful act of infringement, as Mr. Price is believed to have committed, the copyright holder may seek statutory damages of up to $150,000 per offense.

If it can be shown that the value of works which John Price copied illegally exceeds $1000, or that he profited from his actions, he may be charged criminally under section 506 of Title 17. If profit is shown, penalties may include imprisonment of up to 5 years or fines in addition to those imposed civilly. If retail value of the works is used to determine criminal liability penalties may include imprisonment of up to 3 years or additional fines.

2. In order to avoid liability as contributing to the infringement of the copyright of others, one should act in good faith as described in section 512 of Title 17 USC, as amended by the Digital Millennium Copyright Act. Section 512 states that a service provider may not be held liable for the storage, at the direction of users, of any infringing material if the provider has no knowledge or expected knowledge of such infringement, and that "upon obtaining such knowledge or awareness, acts expeditiously to remove, or disable access to, the material."[10] This section is intended to protect ISP's in an effort to foster cooperation in determining subscriber information. However, the definitions set forth in Section 512 do not specify commercial ISP's and define service provider as any "provider of online services or network access, or the operator of facilities therefore." Under this definition anyone who provides equipment and services for accessing the Internet, including private employers, should in theory be covered if they quickly remove access to any infringing material and subsequently notify the offender of such removal.

3. All information to this point will be archived to CD-R and placed in the case file at the state crime lab. This includes the EnCase image file of the submitted floppy found in John Price's computer. The use of EnCase images as best evidence has been upheld in various U.S. court decisions. Several such decisions can be viewed from the website of Guidance Software, Inc. (author of the EnCase forensic suite) at <http://www.encase.com/corporate/legal/index.shtm>.

CD-R is widely expected to last at least 10 years under optimal conditions, but optimal conditions are hard to come by. With proper care, the discs should realistically last at least as long as it would take to bring action against Mr. Price.

4. Unlike copyright violations, the mandatory reporting of distribution of images of child sexual abuse (42 U.S.C. 13032) is specifically limited to service providers accessible by the public at large. Luckily, it is common feeling among the majority of citizens in this country, as well as most others, that child sexual abuse images are an abomination worthy of swift notification to the proper authorities. Since I happen to be employed by

10 Section 512, Title 17, USC. <http://www.copyright.gov/title17/92chap5.htm> (3 March, 2004)

one such authority however, my actions are governed by regulations pertaining to search and seizure.

Had investigator Smith suspected the distribution of child sexual abuse images from the beginning, his natural course of action would have been to contact his local law enforcement agency. They, in turn, would have forwarded the evidence to the lab for analysis or perhaps conducted analysis themselves, if so equipped.

If contraband such as child sexual abuse images were located by me in the course of searching for copyright violations, I would have been required to immediately cease my analysis. At that point an affidavit would need to be submitted requesting a warrant to search for evidence of distribution of child sexual abuse images.

References

Part 1
http://lwn.net/2000/0420/announce.php3
http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html
ftp://ftp.scyld.com/pub/forensic_computing/bmap
http://mail.gnome.org/archives/gtk-app-devel-list/1999-December/msg00271.html
http://www.faqs.org/rfcs/rfc1952.html
http://www.kluge.net/~felicity/ppt/cksum
Special Thanks to Dr. Anton Chuvakin, http://www.netForensics.com

Part 2
http://www.guidancesoftware.com/support/enscript/index.shtm
http://www.vmware.com/products/desktop/ws_features.html
http://www.techspot.com/vb/showthread/t-9798.html
http://www.videocard-forum.com/nvidia (continued)
/How_to_stop_loading_NvCplDaemon_at_startup_403028.html
http://www.liutilities.com/products/wintaskspro/processlibrary
http://www.nerdhelp.com/forums/index.php?act=ST&f=37&t=784&
http://www.annoyances.org/exec/forum/win2000/r1057000914
http://www.safersite.com/pestinfo/w/wildtangent.asp
http://h20015.www2.hp.com/hub_search/document.jhtml?lc=en&docName=bps05762

Part 3
http://www.copyright.gov/title17/92chap5.html
http://www4.law.cornell.edu/uscode/18/2319.html
http://www.encase.com/corporate/legal/index.shtm

Appendix A

Format of each listing is “application name, URL of explanation, explanation.”

hpsysdrv
http://www.techspot.com/vb/showthread/t-9798.html
Krugger wrote:
hpsysdrv.exe;
This item keeps track of how many times the system has been recovered and the times of the first and last recoveries done on the system. Leaving unchecked will sometimes prevent the Keyboard Manager program from detecting that the computer is an HP. Since this program/driver was only made to run on HP, if it can't tell that it is an HP it will not run. If unchecked, it can prevent the running of the Application Recovery CDs, the use of the multimedia keys, and the HP Instant Support. Also seen that without it running, the Riptide Sound card that was installed on some older HP computers stops working.

NvCpl
www.videocard-forum.com/nvidia/How_to_stop_loading_NvCplDaemon_at_startup_403028.html
"Flow" <flowing@zonnet.nl > wrote:
> Why do you want it gone?
> You need it for your desktop,the comp will shutdown if this file is damaged or
> deleted.
> It is for controlpanel\display\advanced\nvidia tabs. That's why it always runs at
> startup.
> It does no harm and will not eat any extra sources.
> In fact if you get error messages about it at startup you need to uninstall
> proper and reinstall your videodrivers.

nwiz
http://www.liutilities.com/products/wintaskspro/processlibrary/nwiz/
Process File: nwiz or nwiz.exe
Process Name: NVIDIA nView Wizard
Description: Application enables user to having 32 virtual desktops, get a desktop larger than the viewable area of the monitor, being able to divide the display across more than one monitor, managing applications and many more functionality.
Company: NVIDIA Corporation
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

KBD
http://www.liutilities.com/products/wintaskspro/processlibrary/kbd/
Process File: kbd or kbd.exe
Process Name: Kbd
Description: Multimedia keyboard manager for logitech keyboards. Required if you use the multimedia keys
Company: Logitech
System Process: No

Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

StorageGuard
http://www.liutilities.com/products/wintaskspro/processlibrary/sgtray/
Process File: sgtray or sgtray.exe
Process Name: StorageGuard Tray Application
Description: System tray bar applicaion which used to remind the user for back up of files.this free utility combines with Backup MyPC , Simple Backup and MS Backup software.
Company: VERITAS Software Corporation.
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

dla
http://www.liutilities.com/products/wintaskspro/processlibrary/tfswctrl/
Process File: tfswctrl or tfswctrl.exe
Process Name: DLA Packet Writing Software
Description: Application used to write data onto CD's directly from Windows applications, without using the actual CD Writing software.
Company: Hewlett-Packard
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

Recguard
http://www.nerdhelp.com/forums/index.php?act=ST&f=37&t=784&
tonya wrote:
x RECGUARD
Y Recguard recguard.exe On HP computers, Recguard prevents the deletion or corruption of the WinXP Recovery Partition. Without it enabled, it is possible to knock that completely out and force the customer to send the PC back to HP for a re-image, possibly at the customer's expense.

IgfxTray
http://www.liutilities.com/products/wintaskspro/processlibrary/igfxtray/
Process File: igfxtray or igfxtray.exe
Process Name: igfxtray
Description: Intel Graphics System Tray icon which gets installed with the drivers for onboard VGA cards based on the Intel 81x graphics chipset. Double-clicking on it enables you to quickly change the display resolution, save your current Display Scheme, or configure your onboard graphics card. You can also configure keyboard hotkeys (shortcuts this is handled by another background task called HKCMD). You can access the same features through the "Intel Graphics Technology" icon in the Control Panel.
Company: Intel Corporation.
System Process: No

Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

HotKeysCmds
http://www.liutilities.com/products/wintaskspro/processlibrary/hkcmd/
Process File: hkcmd or hkcmd.exe
Process Name: Hkcmd
Description: Application which implemements Intel's Hotkey Command.
Company: Intel Corporation
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

PS2
http://www.liutilities.com/products/wintaskspro/processlibrary/ps2/
Process File: ps2 or ps2.exe
Process Name: Ps2
Description: Application provides functionality to keyboard on HP computers.
Company: Hewlett-Packard
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

NAV Agent
http://www.liutilities.com/products/wintaskspro/processlibrary/navapw32/
Process File: navapw32 or navapw32.exe
Process Name: Norton AntiVirus Agent
Description: Background application for Norton AntiVirus which provides Auto-Protection to the system.It runs on Windows 95/98/ME .
Company: Symantec Corporation
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

BJCFD
http://www.annoyances.org/exec/forum/win2000/r1057000914
re: Broadjump programs
Monday, June 30, 2003 at 12:21 pm
Windows 2000 Annoyances Discussion Forum
Posted by Syd:
These programs are often included in the installation software from cable ISPs. They can normally be deleted through Add/Remove Programs.

tgcmd
http://www.liutilities.com/products/wintaskspro/processlibrary/tgcmd/
Process File: tgcmd or tgcmd.exe
Process Name: Tgcmdprovidersbc
Description: Part of a Software from SupportSoft provided to manufacturers (such as

Sony (Vaio Support Agent) and Toshiba (Virtual Tech)) and ISPs (such as Comcast, Cox and Charter (Pipeline Support Agent)) that allows them to offer on-line support. This part ensures that software is installed correctly. Regarded as spyware as it has the ability to retrieve user information.
Company: SupportSoft, Inc
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No
Common Errors: N/A

wcmdmgr, WT GameChannel
http://www.safersite.com/pestinfo/w/wildtangent.asp
Vendor Notes: a web plug in for online gaming, similar to Macromedia's flash. We make and publish 3D online and offline games that play in the web browser. Users are able to purchase our games though our E-Commerce solution and unlock the download version of our games. We also have the WildTangent Web Driver that these games need to run. It is a 3D plug-in for the browser, similar to Macromedia's Flash. We also have a technology called GameChannel that users opt-in to receive. The GameChannel delivers new updates (notifications) to the user's machine when there are new games available to play and purchase.
Category: Adware: Software that brings ads to your computer. Such ads may or may not be targeted, but are "injected" and/or popup, and are not merely displayed within the form of an ad-sponsored application.

checktime
No useful information found, however the online postings of several HP owners indicate that this is a common program installed on HP systems.

Share-to-Web Namespace Daemon
http://h20015.www2.hp.com/hub_search/document.jhtml?lc=en&docName=bps05762
Through direct ties to the HP digital camera, scanner, or all-in-one product, HP's exclusive Share-to-Web software makes it easy to share photos and other digital content through the HPPhoto.com Web site. This Web site provides such services as:
Secure, online photo sharing
Photo reprints
Creating and sending electronic or printed greetings
Online "safe deposit box" storage for important files
Document sharing and collaboration

MSMSGS
http://www.liutilities.com/products/wintaskspro/processlibrary/msmsgs/
Process File: msmsgs or msmsgs.exe
Process Name: MSN Messenger Traybar Process
Description: Tray Bar Icon for MSN Messenger which is an Online Chat and Instant Messaging Client.
Company: Microsoft Corp.
System Process: No
Security Risk (Virus/Trojan/Worm/Adware/Spyware): No

Appendix B

```
/* Windows NT/2000/XP printer spool finder
   Created by modifying the Graphic file Finder released
   by Guidance Software
*/

class MainClass {
  double            Total;
  BookmarkFolderClass Folder;
  SearchClass       Search;
  String            Extension,
                    Path;
  FileClass         Index;
  uint              I;
  int               CHECKUC,
                    CHECKALL,
                    CHECKEXT,
                    CHECKPAGE,
                    ToSearch;


  MainClass() {
    Total = 0;
    I = 0;
    CHECKEXT  = 0;
    CHECKPAGE = 1;
    CHECKUC   = 2;
    CHECKALL  = 3;


Search.Add("\\x01\\x00\\x00\\x00(\\x58[\\x6E\\x00])|(\\x18\\x17)|(\\xD8[\\x10\\x17])|(\\x5c\
\x01)\\x00\\x00", FileClass::GREP);

    Extension = "spl";
    Path      = User.GetExportFolder();
  }

  void Write(EntryClass &entry, long startpos, String name){
    Folder.AddBookmark(entry, startpos, 6, name+ ": " + entry.LongName(),
BookmarkClass::SHOWPICTURE, BookmarkClass::PICTURE);
    I++;
  }


  void CheckEntry(EntryClass &entry) {
    int       result;
    uint      length;
```

```
  ulong     curPos  = 0,
            lastPos = 0,
            startPos;

  bool hit = false;
  FileClass file;
  if (file.Open(entry, FileClass::SLACK)) {
    do {
      User.StatusMessage("Found " + I + " possible spool files");

      if ((result = file.Find(Search, file.GetSize(), length)) >= 0) {
        startPos = file.GetPos();
        hit = true;
        Write(entry, startPos, "spool");
        file.Seek(startPos+6);
      } else
        hit = false;
      if(!hit)
        User.StatusInc(file.GetSize() - lastPos);
      else
        User.StatusInc(file.GetPos() - lastPos);
      lastPos = file.GetPos();
    } while (result >= 0 && (file.GetPos() < file.GetSize()));
  }
}

void Recurse(EntryClass &entry) {
  if ((ToSearch == CHECKALL) && !entry.IsFolder())  {
    CheckEntry(entry);
 } else {
    if ((ToSearch == CHECKEXT) && !Extension.Compare(entry.Extension(), 0))
      CheckEntry(entry);
    if ((ToSearch == CHECKUC) && entry.IsUnallocated())
      CheckEntry(entry);
    if ((ToSearch == CHECKPAGE) && entry.LongName().Compare("PageFile.sys", 0))
      CheckEntry(entry);
  }
  for (EntryClass e = entry.FirstChild(); e; e++)
    Recurse(e);
}

void Count(EntryClass &entry) {
  if (ToSearch == CHECKALL  && !entry.IsFolder())
    Total += entry.LogicalSize();
  else if ((ToSearch == CHECKEXT) && !Extension.Compare(entry.Extension(), 0))
      Total += entry.LogicalSize();
  else if ((ToSearch == CHECKUC) && entry.IsUnallocated())
      Total += entry.LogicalSize();
```

```
   else if ((ToSearch == CHECKPAGE) && entry.LongName().Compare("PageFile.sys",
0))
       Total += entry.LogicalSize();
   for (EntryClass e = entry.FirstChild(); e; e++)
     Count(e);
 }

 void Main(EntryClass case) {
   String BookmarkFolderName    = "Recovered Spool Files",
         BookmarkFolderComment = "Files recovered through EnScript";

   WindowClass dialog(MainWindow,"EnCase Printer Spool file Finder");
     WindowClass page1  (dialog, "Input Parameters");

     page1.AddRadioButtons(10, 20, 220, 0, 0, ToSearch, "Files to search", "Search files
with extension:\tPageFile.sys\tSearch Unallocated Clusters\tSearch all files");
     page1.AddStringEdit (145, 30, 50, 14, WindowClass::AUTOHSCROLL, Extension,
10, 0);
     page1.AddGroupBox  (10, 130, 220, 75, 0, "Output Options");
     page1.AddStaticText(20, 140, 80, 14, 0, "Bookmark Folder Name");
     page1.AddStringEdit(20, 150, 200, 14, WindowClass::AUTOHSCROLL,
BookmarkFolderName, 100, WindowClass::REQUIRED);
     page1.AddStaticText(20, 170, 100, 14, 0, "Bookmark Folder Comment");
     page1.AddStringEdit(20, 180, 200, 14,
WindowClass::AUTOHSCROLL,BookmarkFolderComment, 256, 0);


   if (dialog.Execute() == UserClass::OK){
     Total = 0;
     if ((ToSearch == CHECKEXT) && Extension == "") {
       User.Message(UserClass::MBOK|UserClass::ICONEXCLAMATION,"Error","Must
specify an extension if you wish to search for one");
       return;
     }
     if (case) {
       Folder = User.AddBookmarkFolder(BookmarkFolderName,
BookmarkFolderComment);
       Count(case);
       User.StatusRange("Finding files", Total);
       Recurse(case);
     } else
       User.Message( UserClass::MBOK | UserClass::ICONSTOP, "No open case", "This
script requires an open case!\nPlease open a case, add devices and run the script
again.");
       User.Exit();
   }
 }
}
```

Appendix C

```
/*  Graphics File Finder (v4)
    (c) 2003, Guidance Software, Inc.
    For use with EnCase v.4.16
    Last Modified: January 7, 2004
      * v4.16 required, as well as attendant Include folder
      * Is able to identify footers of AOL ART, GIF, and JPEG files
*/

include "GSI_LogLib"

class FoundGraphicClass {
  String  FormatName;
  long    Offset;
  uint    Length;
}

class MainClass {
  double             Total;
  BookmarkFolderClass Folder;
  SearchClass        Search;
  EntryClass         CurEntry;
  NameListClass      Types,
                     ChosenTypes,
                     ChosenFooters,
                     Headers,
                     Footers;
  String             Extension,
                     Path;
  uint               I,
                     NumHeaders,
                     Section,
                     StackThreshold;
  long               FurthestPos;
  int                CHECKUC,
                     CHECKALL,
                     CHECKEXT,
                     CHECKPAGE,
                     CHECKSEL,
                     ToSearch,
                     SearchCount;
  bool               UseFooters;
  LogClass           Log;

  MainClass() {
    Log.Name = "GFF";
    Log.CurPriority = LogClass::INFO;
```

```
    Total = 0;
    I = 0;
    NumHeaders = 0;
    StackThreshold = 200;
    CHECKEXT  = 0;
    CHECKPAGE = 1;
    CHECKUC   = 2;
    CHECKSEL  = 3;
    CHECKALL  = 4;
    SearchCount = 30015488;

    Add("AOL ART", "\\x4A\\x47\\x04\\x0E\\x00\\x00\\x00\\x00", "\\xCF\\xC7\\xCB");
    Add("BMP", "BM....\\x00\\x00\\x00\\x00....\\x28", "");
    Add("EMF",
"\\x01\\x00\\x00\\x00(\\x58[\\x6E\\x00])|(\\x18\\x17)|(\\xD8\\x10)|(\\x5c\\x01)\\x00\\x00",
"");
    Add("GIF", "GIF8[79]", "\\x00\\x3B");
    Add("JPG", "\\xFF\\xD8\\xFF[\\xFE\\xE0\\xDB]..(EXIF)|(JF(IF)|(XX))", "\\xFF\\xD9");
    Add("Photoshop", "\\x38\\x42\\x50\\x53\\x00\\x01\\x00\\x00\\x00\\x00\\x00\\x00", "");
    Add("TIFF - Big Endian", "\\x4D\\x4D\\x00\\x2A", "");
    Add("TIFF - Little Endian", "\\x49\\x49\\x2a\\x00", "");

    Extension = "doc";
    Path      = User.GetExportFolder();
  }

  void Add(const String &name, const String &header, const String &footer) {
    Types.Add(name);
    Headers.Add(header);
    Footers.Add(footer);
  }

  String GetGrepDescriptions() {
    String ret = "";
    NodeClass grep = Headers.FirstChild();
    for (NodeClass type = Types.FirstChild(); type; type++) {
      ret += type.Name() + ":\t" + grep.Name() + "\n";
      ++grep;
    }
    return ret;
  }

  void InitSearchClass() {
    if (Types.Count() == Headers.Count() && Types.Count() == Footers.Count()) {
      NodeClass type, header, footer;
      for (uint i = 0; i < Types.Count(); ++i) {
        type = Types.GetChild(i);
        header = Headers.GetChild(i);
```

```
      footer = Footers.GetChild(i);
      if (type.IsSelected()) {
        ++NumHeaders;
        Log.Info("Script will look for " + type.Name() + " files");
        Search.Add(header.Name(), FileClass::GREP);
        ChosenTypes.Add(type.Name());
      }
    }
    if (0 < NumHeaders) {
      for (uint i = 0; i < Types.Count(); ++i) {
        type = Types.GetChild(i);
        footer = Footers.GetChild(i);
        if (type.IsSelected()) {
          if (UseFooters && "" != footer.Name()) {
            ChosenFooters.Add(footer.Name());
            Search.Add(footer.Name(), FileClass::GREP);
          }
          else {
            ChosenFooters.Add("");
          }
        }
      }
    }
    else {
      User.Message(UserClass::MBOK|UserClass::ICONEXCLAMATION,"Error","You
must select at least one graphics format!");
      User.Exit();
    }
    for (type = Search.FirstChild(); type; type++) {
      Log.Debug("Search term: " + type.Name());
    }
  }
  else {
    User.Message(0, "Error", "Number of Headers does not match number of available
graphic types!");
    User.Exit();
  }
 }

 void Write(EntryClass &entry, const FoundGraphicClass &hit) {
  ++I;
  if (hit.Offset + hit.Length > FurthestPos)
    Log.Debug("New Furthest Pos: " + (hit.Offset + hit.Length));
  FurthestPos = Max(FurthestPos, hit.Offset + hit.Length);
  Log.Debug("Hit: " + entry.Name() + ", " + hit.FormatName + ", " + hit.Offset + ", " +
hit.Length + ", FurthestPos = " + FurthestPos);
  String comment = hit.FormatName + ":\t" + entry.Name() + "\nFile offset:\t" +
hit.Offset + "\nLength:\t" + hit.Length + "\n";
```

```
    Folder.AddBookmark(entry, hit.Offset, hit.Length, comment,
BookmarkClass::SHOWPICTURE, BookmarkClass::PICTURE);
  }

  long Min(long x, long y) {
    if (x < y)
      return x;
    else
      return y;
  }

  long Max(long x, long y) {
    if (x > y)
      return x;
    else
      return y;
  }

  long GetSearchCount(FileClass &file) {
    if (file.GetPos() / SearchCount == Section) {
      return -1;
    }
    else {
      ++Section;
      return SearchCount;
    }
  }

  bool FindGraphic(FileClass &file, FoundGraphicClass &hit, int lastResult, uint
stackLevel) {
    int result;
    long count    = GetSearchCount(file),
         startPos = file.GetPos();
    ScopedLogClass scope(Log, "FindGraphic (" + lastResult + ")");
    Log.Debug("Searching from " + startPos + " for " + count + " bytes");
    if ((result = file.Find(Search, count, hit.Length)) >= 0) {
      hit.Offset = file.GetPos();
      file.Skip(hit.Length);
      Log.Debug("Result = " + result + ", Pos = " + hit.Offset);
      if (result < NumHeaders) {
        hit.FormatName   = ChosenTypes.GetChild(result).Name();
        NodeClass footer = ChosenFooters.GetChild(result);
        if ("" != footer.Name() && stackLevel < StackThreshold) {
          do {
            FoundGraphicClass nexthit;
            bool success = FindGraphic(file, nexthit, result, stackLevel + 1);
            if (success) {
              if (hit.FormatName == nexthit.FormatName) {
```

```
            Write(CurEntry, nexthit);
            success = false;
          }
          else if ("" == nexthit.FormatName)
            hit.Length = nexthit.Offset + nexthit.Length - hit.Offset;
        }
        file.Seek(Max(FurthestPos, nexthit.Offset + nexthit.Length));
      } while (!success && FileClass::EOF != file.Peek());
    }
    return true;
  }
  else {
    hit.FormatName = "";
    return (-1 < lastResult && lastResult < NumHeaders &&
Search.GetChild(result).Name() == ChosenFooters.GetChild(lastResult).Name());
  }
 }
 else {
  long max   = Min(file.GetSize(), SearchCount * (Section + 1));
  hit.Length = max - startPos;
  hit.Offset = startPos;
  hit.FormatName = "";
  return false;
 }
}

void SearchFile(FileClass &file) {
 FoundGraphicClass hit;
 long              startPos;

 file.Seek(0);
 FurthestPos = 0;

 while (FileClass::EOF != file.Peek()) {
  startPos = file.GetPos();
  if (FindGraphic(file, hit, -1, 0)) {
   Write(CurEntry, hit);
  }
  User.StatusMessage("Found " + I + " pictures");
  file.Seek(Max(FurthestPos, hit.Offset + hit.Length));
  User.StatusInc(file.GetPos() - startPos);
 }
}

void CheckEntry(EntryClass &entry) {
 FileClass file;
 if (file.Open(entry, FileClass::SLACK)) {
  Log.Debug("Examining entry: " + entry.FullPath());
```

```
      CurEntry = entry;
      Section = -1;
      SearchFile(file);
    }
  }

  bool Filter(EntryClass &entry) {
    if (!entry.IsFolder()) {
      if (CHECKALL == ToSearch)
        return true;
      else if (CHECKEXT == ToSearch && Extension == entry.Extension())
        return true;
      else if (CHECKUC == ToSearch && entry.IsUnallocated())
        return true;
      else if (CHECKPAGE == ToSearch && (0 ==
"PageFile.sys".Compare(entry.Name(), false) || 0 ==
"win386.swp".Compare(entry.Name(), false)))
        return true;
      else if (CHECKSEL == ToSearch && entry.IsSelected())
        return true;
    }
    return false;
  }

  void Recurse(EntryClass &entry) {
    if (Filter(entry))  {
      CheckEntry(entry);
    }
    for (EntryClass e = entry.FirstChild(); e; e++)
      Recurse(e);
  }

  void Count(EntryClass &entry) {
    if (Filter(entry)) {
      Total += entry.PhysicalSize();
    }
    for (EntryClass e = entry.FirstChild(); e; e++)
      Count(e);
  }

  void Main(EntryClass case) {
    User.ClearConsole();

    String BookmarkFolderName    = "Recovered Graphics Files",
           BookmarkFolderComment = "Files recovered";

    WindowClass dialog(MainWindow,"EnCase Graphic file Finder");
      WindowClass page1  (dialog, "Input Parameters");
```

```
    WindowClass page2  (dialog, "     Help       ");

    page1.AddListBox     (10, 10, 220, 40, 0, "Graphic Types", Types,
WindowClass::LISTCHECK);
    ToSearch = 4;
    page1.AddRadioButtons(10, 70, 220, 0, 0, ToSearch, "Files to search", "Files with
extension:\tPageFile.sys\tUnallocated Clusters\tSelected files only\tAll files");
    page1.AddStringEdit (125, 80, 50, 14, WindowClass::AUTOHSCROLL, Extension,
10, 0);
    page1.AddCheckBox  (10, 170, 0, 0, 0, "Use footer analysis (ART, GIF, JPEG)",
UseFooters);
    page1.AddGroupBox  (10, 185, 220, 75, 0, "Output Options");
    page1.AddStaticText(20, 195, 80, 14, 0, "Bookmark Folder Name");
    page1.AddStringEdit(20, 205, 200, 14, WindowClass::AUTOHSCROLL,
BookmarkFolderName, 100, WindowClass::REQUIRED);
    page1.AddStaticText(20, 225, 100, 14, 0, "Bookmark Folder Comment");
    page1.AddStringEdit(20, 235, 200, 14,
WindowClass::AUTOHSCROLL,BookmarkFolderComment, 256, 0);

    page2.AddGroupBox  (10, 10, 220, 30, 0, "Graphics File Finder EnScript");
    page2.AddStaticText(20, 20, 0, 0, 0,    "Locate various graphics files\n  and
bookmark them.");
    page2.AddGroupBox  (10, 50, 220, 50, 0, "Files to search");
    page2.AddStaticText(20, 60, 0, 0, 0,    "Extension:\tonly searches files with the
ext.\n"
                                "PageFile.sys:\tonly search the pagefile.sys.\n"
                                "Unallocated:\tonly searches unallocated clusters\n"
                                "All:\t\tsearches every file.");
    page2.AddGroupBox  (10, 110, 360, 85, 0, "Graphics to search - (Headers)");
    page2.AddStaticText(20, 120, 0, 0, 0, GetGrepDescriptions());
    page2.AddStaticText(10, 200, 0, 0, 0,"Please email any bugs or requests
concerning this EnScript\nto EnScript@encase.com at Guidance Software, Inc.");

   if (dialog.Execute() == UserClass::OK){
    Total = 0;
    if ((ToSearch == CHECKEXT) && Extension == "") {
      User.Message(UserClass::MBOK|UserClass::ICONEXCLAMATION,"Error","Must
specify an extension if you wish to search for one");
      return;
    }
    if (case) {
      DateClass now;
      now.Now();
      uint start = now.GetUnix();
      LogRecordClass runLog("Run Log");
      Log.RunLog = runLog;
      Log.OutputToRunLog = true;
      Log.Info("Script started");
```

```
        String config = "Script will look for graphics in ";
        if (CHECKEXT == ToSearch)
          config += "files with " + Extension + " as their extension";
        else if (CHECKUC == ToSearch)
          config += "unallocated space";
        else if (CHECKPAGE == ToSearch)
          config += "swap space";
        else if (CHECKSEL == ToSearch)
          config += "selected files only";
        else if (CHECKALL == ToSearch)
          config += "all files";
        Log.Info(config);
        InitSearchClass();
        Folder = User.AddBookmarkFolder(BookmarkFolderName,
BookmarkFolderComment);
        Count(case);
        Log.Info("Script will search " + Total + " bytes");
        User.StatusRange("Finding images", Total);
        Recurse(case);
        Log.Info("Script found " + I + " graphics");
        now.Now();
        Log.Info("Script completed in " + (now.GetUnix() - start) + " seconds");
        Folder.AddDatamark("Run Log", runLog);
      }
      else {
        User.Message( UserClass::MBOK | UserClass::ICONSTOP, "No open case", "This
script requires an open case!\nPlease open a case, add devices and run the script
again.");
      }
    }
  }
}
```

Appendix D

Attaining 100% certainty in any aspect of computer forensics is generally believed to be a practical impossibility. There is always an alternative explanation for any scenario, no matter how unlikely it may be. As discussed briefly in the above section titled “Deleted Files” the recovery of files on an NTFS volume requires reference to deleted entries in the Master File Table. The MFT has to be assumed reliable, but as with most things in life it is not foolproof. Here’s a brief experiment I conducted:

I first created a directory called “parent” off of the root directory and added 5 random files. I then previewed the drive in EnCase and saw the correct listing of directories and files, with an indication that “parent” had MFT record # 9743. I next deleted the 5 files from parent and then previewed the drive again. “Parent” was shown to contain 5 files each marked as deleted. I then deleted “parent” and again previewed the drive. “Parent” was now also marked as deleted and still contained 5 deleted files. I sorted the list by MFT# and filtered so that only the deleted files would show. “Parent” was the first deleted file (lowest MFT #) so its record should get used first by any new files. Had “parent” been at the end of the list of deleted files this would have been a lot more work. I copied a new file to the volume and once more previewed the drive in EnCase. This time “parent” was gone, and the 5 files were now listed under "Lost Files". My newly copied file had been given MFT# 9743. I deleted the new file, and created a new directory called “fake_parent” off of the root directory. I previewed the drive one last time and observed that “fake_parent” now had MFT# 9743 and was said to contain my original 5 files, each still marked as deleted!

As stated in the “Deleted Files” section, since all 5 files listed MFT record 9743 as that of their parent, and 9743 contained a valid directory, EnCase had to assume that they must have all been deleted from “fake_parent.”