GIAC Certified Forensic Analyst (GCFA) Practical Assignment Version 1.2 (December 30, 2002)

William (BJ) Bellamy Jr. GSEC, GCIH

Abstract

This paper addresses the three-part requirements of the GCFA Practical Assignment Version 1.2.

The first section is the analysis of an unknown binary file provided by GIAC for this exercise. Several characteristics of the file and its operation are to be identified.

The second section is an investigation of a system acquired from an organization's department of surplus properties. The purpose was to determine if surplus systems, specifically personal computers DASD (Direct Access Storage Device) were properly sanitized before releasing control of them to the general public. If not properly sanitized, they would be inspected for confidential, proprietary, or otherwise inappropriately information.

The third section looks at the legal considerations of an incident handling scenario involving and ISP and law enforcement officials.

Part 1 – Analyze an unknown binary

In this section the focus is on collecting information about an unknown binary program file. As with many technicians, I find myself building assumptions and conclusions with each new piece of information found. Each new clue or detail reshapes the completed picture I have already built; leaving the final conclusion based on just the first few findings. While this approach is helpful in some types of troubleshooting and investigation, it is consistently misleading in digital forensics.

So, this "evidence collection" phase was particularly helpful in practicing to not make any conclusions or assumptions, but simply to gather raw facts for later evaluation/analysis. The objective is to take notice of items that will to the collection of information in areas where you would not normally look.

In this case, since the subject file is an unknown binary, tools like objdump, strace, and ldd may point to files, devices, or programmatic functions not otherwise considered. If the subject of investigation were a disk image harvested with dd, then file system tools would be used to collect the material to be analyzed.

Sanitizing DASD

One of the basic steps in forensic analysis is to sanitize the analyst's disks (destructively overwriting any previously store material) so that the suspect materials being analyzed, and the subsequent findings, are not contaminated.

However, in this particular project, the assumption was that recommendations for effective DASD sanitation would need to be developed for systems owners so they could reduce the risk of leaking confidential or proprietary digital material through surplus systems.

The document describing the resulting recommendations for sanitizing DASD is included as an appendix to this paper.

As the sanitizing paper was developed, it was used to prepare the disks used in this analysis hosts. Rather than devote space here to describe the process, please refer to the appendix for details.

The Analysis Environment

To prepare to analyze the unknown binaries, first three hosts were configured, one for each of the following task:

1. Analysis and investigation
2. Monitoring the lab network
3. Harvesting the suspect disk image

The analysis host would be used to perform the analysis and reporting tasks throughout the project. This system is a dual-boot laptop, with a fresh install of Redhat 8.0 [(1)] and MS Windows XP [(2)]. The Linux partitions were formatted as ext2fs [(3)] so that ext2fs drivers could be used to provide read-only access from the Windows side to the Linux partitions. The ntfs drivers from the ntfs project [(4)] were used to provide the Linux side with read-only access to the Windows C:\ ntfs drive.

Throughout this analysis project, the root account will be used while in Linux, and the local Administrator account while in Windows. This will allow maximum access and control of the system. While this also exposes the system to malicious activity by this unknown binary, and other attack vectors, sufficient care will be taken to isolate the material being analyzed and to protect the integrity of the analysis host.

As you can see in the illustration below, the partition table reports that this single hard disk contains 5 partitions. The first (hda1) is the Windows ntfs partition (the C:\ drive). When running under Linux, the /dev/hda1 device is mounted with an ntfs file system driver. The third partition (/dev/hda3) is the root directory for Linux. The Windows ext2 driver referees to it as drive L, partition 3. The fourth is the extended partition record used to overcome the 4-partition limit of the partition table. The 31 blocks (522112-522081=31) represent the area containing the extended partition information. The remaining space is allocated as the Linux swap partition.

```
Device      Boot  Start    End    #cyls    #blocks    Id    System
/dev/hda1   *        0+   1274    1275-      10241   406  HPFS/NTFS
/dev/hda2         1275    1287       13     104422+   83      Linux
/dev/hda3         1288    3582     2295   18434587+   83      Linux
/dev/hda4         3583    3647       65     522112+    f  Win95 Ext'd (LBA)
/dev/hda5         3583+   3647       65-    522081    82    Linux swap

3648 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0
```

I have worked with DOS and Windows for many years, but am relatively new to Linux. I chose to use a dual-boot system for several reasons, including:

- to provide the widest range of tools;
- tools used in one operating system would help validate the findings of tools used in the other operating system; and
- as an opportunity to learn more about Linux, which is clearly a superior platform for forensic analysis and other information security tasks.

The monitoring host was a second laptop configured with a fresh install of Redhat 7.2. This host was used to observe the traffic on the lab network, to observe the network activity of this unknown binary and to watch for signs of any other unexpected activity. The monitoring system was also used to store copies of the material being analyzed and the analysis findings in order to better manage the limited disk space on the analysis host.

The disk-imaging host was used for several tasks. First, it was obviously used to read the suspect hard disk and create the bit-by-bit copy that was actually used during the analysis. Second, this host will be used to execute the unknown binary for observation. And third, it was used to run long-term, CPU-intensive tasks such as password cracking and steganography analysis, freeing the analysis host for reporting and manual analysis tasks.

All three hosts were placed on a stand-alone 10/100 Ethernet network hub. Neither the hub nor hosts were connected to any other network or system from the point their operating systems were installed (from the original CDs). The unknown binary was transferred from an Internet connected PC onto the analysis host via floppy diskette.

After the analysis was complete, the analysis host was connected to the Internet and used to integrate the analysis findings and research material into this final paper.

The analysis host	Gateway 1450SE laptop • 512 MB RAM • 1.33 GHz Celeron CPU • Hard disk 20 GB • 10/100 Ethernet adapter • CDROM 4x • Operating Systems Redhat Linux 8.0 with no updated or patches applied. All packages were installed. Microsoft Windows XP home edition.

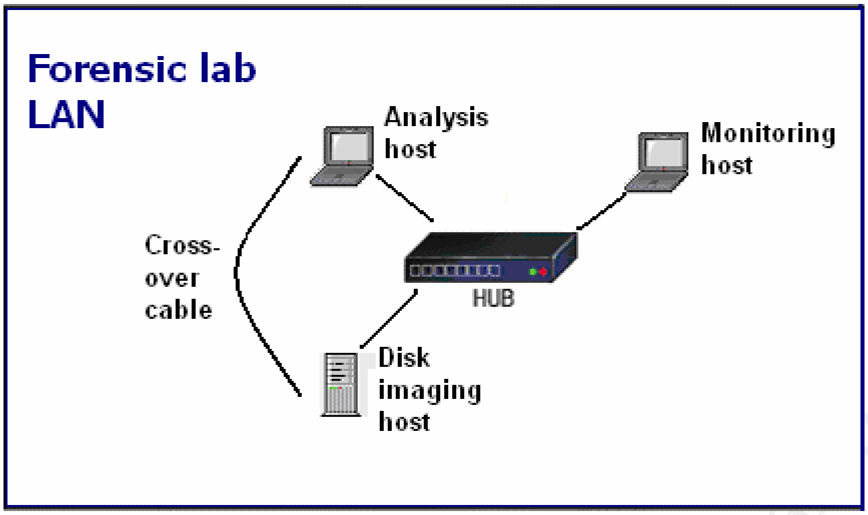
	Service pack 1 applied.
The monitoring host	Gateway 9550 laptop • 512 MB RAM • 700 MHz Pentium processor • Hardisk 27 GB • 10/100 Ethernet adapter • CDROM 4x • Operating systems Redhat Linux 8.0 with no updates or patches applied. All packages were installed. Microsoft Windows 2000 Professional. All available patches and service packs applied.
The disk-imaging host	Gateway 9300 tower • 128 MB RAM • 600 MHz Pentium processor • Hard disk 2.0 GB • CDROM 4x • Operating system Redhat Linux 7.3 and Trinux • 10/100 Ethernet adapter
10/100 network hub	SMC EtherEZ Hub 3605T 4 standard ports 2 ports for chaining (not used in this investigation)

A separate system was used to capture an identical copy of the subject hard disks. The requirement was to make a bit-for-bit identical copy to use in the actual analysis. Actually, the original hard drives and the initial copy were all secured and stored in a locked area, and only a copy of the initial copy was used to create yet another copy on which the analysis was actually performed. This provided redundant safety nets in case the copy being worked with became corrupt or tainted and had to be replaced.

It was necessary to have read-only access to the original hard drive only once. If one of the copies being worked with became corrupted or for some reason needed to be replaced, a fresh copy could be produced from the second-generation copy without having to use either the original hard disk or the originally harvested image.

Laboratory Layout

All three hosts were connected through a 4-port 10/100 Ethernet hub. A hub was used rather than a network switch so that monitoring (sniffing) the lab network would be easier. Network switches create virtual sessions directly between the two participating hosts, bypassing the cable runs to all other hosts.



Physically and logically the lab LAN was isolated, or air-gapped, from any other systems. There were no physical (cable) connections from any of these devices to any other network or communications system. The lab LAN was setup in a low traffic, locked room and all systems were powered down when not actively involved in the analysis process.

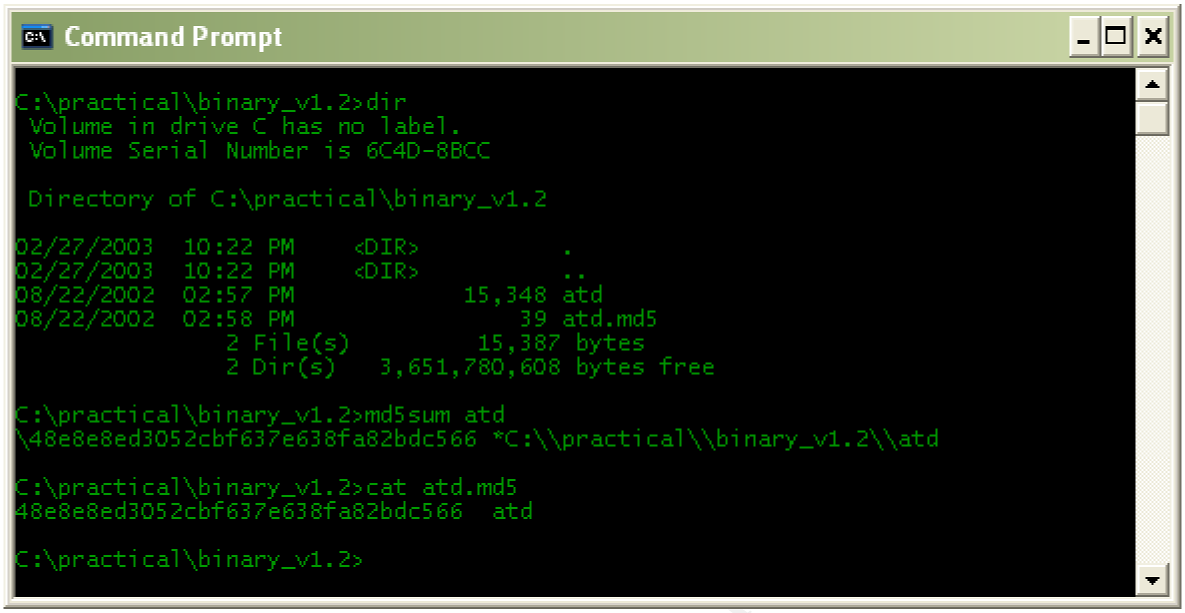
A single crossover cable was used at times to directly connect two hosts together for file transfers, backups.

Binary Details

It is critical that the newly installed analysis host not be exposed to other hosts, either through a LAN or the Internet. The zipped file containing the unknown binary was downloaded directly to a new floppy diskette using a neutral host equipped with up-to-date virus protection and host based firewall software.

The zip file was then unzipped [10] from the floppy into a new folder (C:\practical) on the analysis host. The md5 (Message Digest version 5) hash of both unzipped files were immediately calculated using md5sum [11]. The floppy was then labeled and locked, using its read-only tab, and stored for later use if necessary. This floppy was treated as the original media.

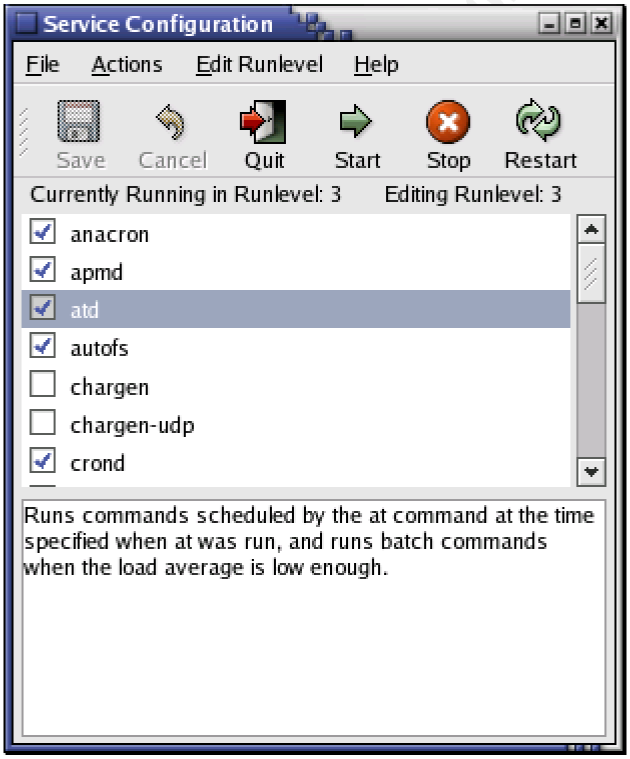
This zip archive contained 2 files, atd and atd.md5. The atd.md5 file contains the md5 hash value of the atd file. By recalculating the md5 hash of the atd file and comparing it to the hash value in atd.md5 the unzipped copy was shown to be identical to the original zipped copy.



```
C:\practical\binary_v1.2>dir
 Volume in drive C has no label.
 Volume Serial Number is 6C4D-8BCC

 Directory of C:\practical\binary_v1.2

02/27/2003  10:22 PM    <DIR>          .
02/27/2003  10:22 PM    <DIR>          ..
08/22/2002  02:57 PM            15,348 atd
08/22/2002  02:58 PM                39 atd.md5
               2 File(s)         15,387 bytes
               2 Dir(s)   3,651,780,608 bytes free

C:\practical\binary_v1.2>md5sum atd
\48e8e8ed3052cbf637e638fa82bdc566 *C:\\practical\\binary_v1.2\\atd

C:\practical\binary_v1.2>cat atd.md5
48e8e8ed3052cbf637e638fa82bdc566  atd

C:\practical\binary_v1.2>
```

In the Windows world some tools and techniques exist for analyzing ELF (Executable Linkable Format; the binary format for Linux executable programs) binaries. However, switching to Linux provides several more tools that would help to analyze this ELF binary file, atd.

At first glance, this appeared to be the stock atd binary, or "at daemon," which is a job scheduler similar to the "at" command in Windows, though crond is more often used today. So one obvious question is, "is this a trojaned version of crond?"



Currently Running in Runlevel: 3   Editing Runlevel: 3

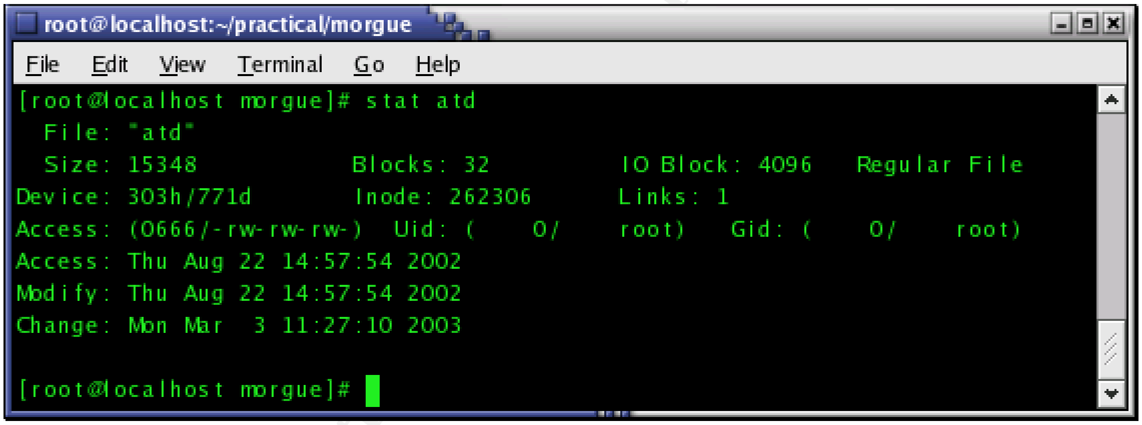
☑ anacron
☑ apmd
☑ atd
☑ autofs
☐ chargen
☐ chargen-udp
☑ crond

Runs commands scheduled by the at command at the time specified when at was run, and runs batch commands when the load average is low enough.

```
L:\etc\rc.d\init.d>ls atd -o
-r--r--r--   1 user          1176 Jul 24  2002 atd

L:\root\practical\atd\bin>ls atd -o
-r--r--r--   1 user         15348 Aug 22  2002 atd
```

The "real" atd daemon checks in at 1176 bytes, while this unknown binary is 15,348. If they are related, the unknown file is 15 times larger than the known. If this is the "at" daemon, it will soon become apparent. If not, then it is likely that the unknown binary has been renamed to atd as a stealth measure. A file named atd would not catch an administrator's attention since it would be expected to be located in a privileged folder with other binaries. If the file name is a stealth technique, it also indicates the possible level of sophistication and familiarity with *nix operating systems of the person who renamed or planted this file.

The stat tool reports the file's mode/permissions, along with confirming the MAC times. In this case, the owner and group is root, and all accounts have Read-Write permission to this file. The MAC times indicate the file was last modified on Thu Aug 22, though the date last changed is the date this file was placed on this system.



```
[root@localhost morgue]# stat atd
  File: "atd"
  Size: 15348           Blocks: 32          IO Block: 4096    Regular File
Device: 303h/771d       Inode: 262306       Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Aug 22 14:57:54 2002
Modify: Thu Aug 22 14:57:54 2002
Change: Mon Mar  3 11:27:10 2003

[root@localhost morgue]#
```

The file tool was then used to determine the type of contents in the atd file. In the Windows world, file types are often determined by their three-character name suffix. Similar to TCP/IP port assignments, there is no functional connection between the cosmetic name suffix and the file's actual content or format. There is nothing to hinder running a web service on port 25 and an email on 80. The world has simply consented to a common list of assignments. The same is true for Windows/DOS files. I can name a binary file .doc and still execute it.

In the example below, the Windows-based whoami.exe program is copied to a new filename. The effect is to change this executable file's name suffix from .exe to .txt. Afterwards, the .exe version has been deleted and the .doc version is executed with no problems.

```
C:\>dir whoami.exe
11/11/1999  01:00 AM             7,680 whoami.exe

C:\>xcopy whoami.exe whoami.doc
1 File(s) copied
```

```
C:>dir whoami*
11/11/1999  01:00 AM             7,680 whoami.doc
11/11/1999  01:00 AM             7,680 whoami.exe

C:\>del whoami.exe

C:\>whoami.doc
Owner
```

The file tool, however, directly interrogates different parts and characteristics of the target file, looking for "magic" signatures and other characteristics that clearly classify the content of the file rather than simply imply the content, as with the filename suffix in the Windows world.

The Linux file (16) stat (17) and ldd (18) tools provide useful information about this file.

The file tool identifies atd as an ELF binary. Specifically, this is a 32-bit ELF executable binary. The byte layout is identified as LSB, or Least Significant Bit. This executable has been built for the 80386 Intel processors.

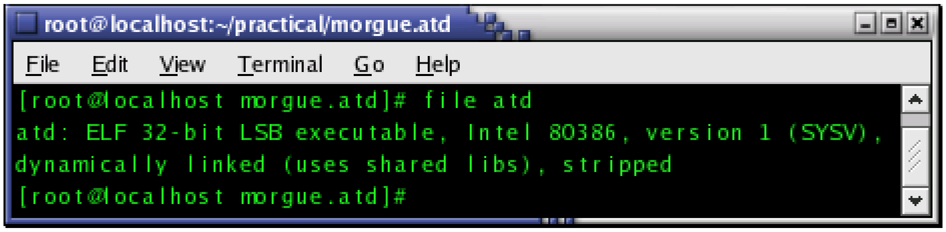
This executable is dynamically linked, meaning that it relies on the availability of specific code libraries for some portion of its operation. In the Windows world, DLL (Dynamically Linked Library) files store code segments that programs can reference during execution as needed. In the Linux world, shared libraries perform the same function.

One of the advantages of dynamically linking a program, which is an option at compile time, is a much smaller executable file is built. Of course, the binary must work from the assumption the libraries of code segments it requires will, in fact, be available.

Another advantage is the consolidation of function code. Not only can each unrelated program rely on the one copy of code stored in the DLL or shared lib, but, as updates to the libraries occur, the update is effectively applied to all programs that will eventually call the function.

Finally, the file tool reports that atd is "stripped." Normally, a binary contains header and debug information that is helpful when first building or debugging a program. This header and debug information can include variable names, comments, version information, and other structural and descriptive information.

So, it appears that the goal was to produce the smallest version possible of the program, making the assumption that all required code libraries would actually be available on whichever host(s) it is executed. The stripping not only helps reduce the file size, but also removes much of the descriptive information that would help determine its purpose and origin.


root@localhost:~/practical/morgue.atd
File Edit View Terminal Go Help

```
[root@localhost morgue.atd]# file atd
atd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), stripped
[root@localhost morgue.atd]#
```

The readelf tool was used next to interrogate the ELF binary, providing a large amount of information about the structure and content of the compiled executable.

The -a parameter instructs readelf to report on each section found in the target ELF binary. This is one instance where a functional familiarity with the C programming language would be critically helpful. Lacking that familiarity, I have to rely on the obvious items like keywords. In this case, I first culled out those keywords I thought relevant, and researched them through the Linux man (manual) pages.

```
# readelf -a -n atd

ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048db0

Relocation section '.rel.plt' at offset 0x8dc contains 50 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
(Several lines deleted for readability)
0804c590  00000607 R_386_JUMP_SLOT   08048ad8   geteuid
0804c594  00000807 R_386_JUMP_SLOT   08048ae8   getprotobynumber
0804c5a4  00000d07 R_386_JUMP_SLOT   08048b28   getpid
0804c5b4  00001207 R_386_JUMP_SLOT   08048b68   getuid
0804c5bc  00001507 R_386_JUMP_SLOT   08048b88   socket
0804c5d4  00001e07 R_386_JUMP_SLOT   08048be8   inet_addr
0804c5d8  00001f07 R_386_JUMP_SLOT   08048bf8   chdir
0804c5e0  00002107 R_386_JUMP_SLOT   08048c18   setsockopt
0804c5ec  00002507 R_386_JUMP_SLOT   08048c48   umask
0804c5f0  00002607 R_386_JUMP_SLOT   08048c58   signal
0804c610  00002e07 R_386_JUMP_SLOT   08048cd8   inet_ntoa
0804c614  00002f07 R_386_JUMP_SLOT   08048ce8   getppid
0804c61c  00003107 R_386_JUMP_SLOT   08048d08   gethostbyname
0804c63c  00003b07 R_386_JUMP_SLOT   08048d88   setsid
```

The descriptions of the keywords found by readelf begin to paint a picture of the nature and use of this unknown binary.

Keyword	Description

geteuid	Get the effective user ID of this process, that is the user ID of the set ID bit associated with the file being run.
getprotobynumber	Rather than the common name of a given protocol (ftp, telnet, SMTP) the protocol's number is returned. For example; 21/22 for ftp, 23 for telnet, and 25 for SMTP.
getpid	This call returns the process ID (pid) of the current process. It is a program in execution asking the operating system "what process number am I?"
getuid	This call returns the real user ID associated with the current process. It is a program in execution asking the operating system "Who launched me into execution?"
socket	The socket call creates an endpoint for communications. Sockets are the logical components at both the sender and the receiver into which the transport (on the wire) protocol is plugged in. Sockets can create these communications between two processes running on the same host or on two separate hosts.
chdir	This call simply changes the current working directory within a file system.
setsockopt	This call allows the program to manipulate or report the options of a specific socket.
signal	This C call creates a new signal handler for a specific signal. The signal handler responds to the arrival of a signal with the associated signal number, the handler responds by performing a predefined response task.
inet_ntoa	This call converts the host address from its native network byte order to a standard number in dot notation.
getppid	This call retrieves the process ID of the parent (hence ppid, parent process ID) of the current process.
gethostbyname	This call queries the DNS infrastructure to retrieve the name of a given host network address.
setsid	This call launches a program in a new session.

These keywords would seem to indicate that the atd binary provides some sort of network service. The program is preoccupied with local system accounts and processes, along with basic networking tasks. The calls to network functions paint a picture of a program that intended to wait for an event (using **signal**), and that takes a close look at network packets (using setsockopt, inet_ntoa, getprotobynumber, socket). The "d" in at**d** implies that the program is a daemon (service), and these network calls support that assumption, though they do not prove it.

In the illustration below, objdump is run to further interrogate this unknown ELF binary. The –x parameter instructs objdump to report on all section headers found in this compiled binary. The –s parameter additionally requests the full contents of each section be displayed in the common hex/ASCII format.

As with most of these type tools, the output can be excessively large. So, here again, all but the relevant lines have been omitted, and the items of particular interest **bolded**.

```
# objdump -x -s atd
```

The atd file is a 32-bit ELF program compiled for the Intel 386 family of processors.

```
atd:     file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x08048db0

Dynamic Section:
  NEEDED      libc.so.5
```

Going back to our executable, the .interp section simply contains an ASCII string that is the name of the dynamic loader. Currently this will always be /lib/elf/ld-linux.so.1 (the dynamic loader itself is also an ELF shared library). (57 Youngdale)

```
Contents of section .interp:
 80480d4 2f6c6962 2f6c642d 6c696e75 782e736f  /lib/ld-linux.so
 80480e4 2e3100                               .1.
```

Next you will notice 3 sections, names .hash, .dynsym, and .dynstr. This is a minimal symbol table used by the dynamic linker when performing relocations. You will notice that these sections are mapped into virtual memory (the virtual address field is non-zero). At the very end of the image are the regular symbol and string tables, and these are not mapped into virtual memory by the loader. The .hash section is just a hash table that is used so that we can quickly locate a given symbol in the .dynsym section, thereby avoiding a linear search of the symbol table. A given symbol can typically be located in one or two tries through the use of the hash table. (57 Youngdale)

```
Contents of section .dynstr:
 80486ac 006c6962 632e736f 2e35006c 6f6e676a  .libc.so.5.longj
 804875c 736f636b 6574005f 5f656e76 69726f6e  socket.__environ
 804876c 00627a65 726f005f 696e6974 00616c61  .bzero._init.ala
 804877c 726d005f 5f6c6962 635f696e 69740065  rm.__libc_init.e
 804879c 696c6c00 696e6574 5f616464 72006368  ill.inet_addr.ch
 804882c 67657468 6f737462 796e616d 65005f66  gethostbyname._f
```

The .rodata code segment is allows to contain constant pointers and strings. (58 Bierne)

```
Contents of section .rodata:
 804a8e8 0a6c6f6b 69643a20 436c6965 6e742064  .lokid: Client d
 804a8f8 61746162 61736520 66756c6c 00444542  atabase full.DEB
 804a918 6e6f6e6f 0a00322e 30000a6c 6f6b6964  nono..2.0..lokid
 804a928 20766572 73696f6e 3a090925 730a0072   version:..%s..r
 804a938 656d6f74 6520696e 74657266 6163653a  emote interface:
 804a958 73706f72 743a0925 730a0058 4f520061  sport:.%s..XOR.a
 804a968 63746976 65206372 7970746f 67726170  ctive cryptograp
 804a978 68793a09 25730a00 73657276 65722075  hy:.%s..server u
 804a988 7074696d 653a0909 252e3032 66206d69  ptime:..%.02f mi
 804a998 6e757465 730a0063 6c69656e 74204944  nutes..client ID
 804a9a8 3a090925 640a0070 61636b65 74732077  :..%d..packets w
 804a9b8 72697474 656e3a09 256c640a 00627974  ritten:.%ld..byt
 804ab48 5d206e61 6d65206c 6f6f6b75 70206661  ] name lookup fa
 804ab58 696c6564 00000000 00000000 00000000  iled............
 804af98 66617461 6c5d2043 616e6e6f 7420676f  fatal] Cannot go
 804afa8 20646165 6d6f6e00 5b666174 616c5d20   daemon.[fatal]
 804afb8 43616e6e 6f742063 72656174 65207365  Cannot create se
 804afc8 7373696f 6e002f64 65762f74 7479005b  ssion./dev/tty.[
 804afd8 66617461 6c5d2063 616e6e6f 74206465  fatal] cannot de
 804afe8 74616368 2066726f 6d20636f 6e74726f  tach from contro
 804aff8 6c6c696e 67207465 726d696e 616c002f  lling terminal./
 804b088 6c6f6b69 64202d70 2028697c 7529205b  lokid -p (i|u) [
 804b098 202d7620 28307c31 29205d0a 005b6661   -v (0|1) ]..[fa
 804b0a8 74616c5d 20736f63 6b657420 616c6c6f  tal] socket allo
 804b0b8 63617469 6f6e2065 72726f72 005b6661  cation error.[fa
 804b0c8 74616c5d 2063616e 6e6f7420 63617463  tal] cannot catc
 804b0d8 68205349 47555352 31004361 6e6e6f74  h SIGUSR1.Cannot
 804b0e8 20736574 2049505f 48445249 4e434c20   set IP_HDRINCL
 804b0f8 736f636b 6574206f 7074696f 6e005b66  socket option.[f
 804b128 74283229 000a4c4f 4b493209 726f7574  t(2)..LOKI2.rout
 804b138 65205b28 63292031 39393720 6775696c  e [(c) 1997 guil
 804b148 6420636f 72706f72 6174696f 6e20776f  d corporation wo
 804b158 726c6477 6964655d 0a005b66 6174616c  rldwide]..[fatal
 804b1d8 6e672065 72726f72 000a6c6f 6b69643a  ng error..lokid:
 804b1e8 20736572 76657220 69732063 75727265   server is curre
 804b1f8 6e746c79 20617420 63617061 63697479  ntly at capacity
 804b208 2e202054 72792061 6761696e 206c6174  .  Try again lat
 804b218 65720a00 0a6c6f6b 69643a20 43616e6e  er...lokid: Cann
(Lines omitted for readability)
 804b3e8 6974000a 6c6f6b69 643a2063 616e6e6f  it..lokid: canno
 804b3f8 74206c6f 63617465 20636c69 656e7420  t locate client
 804b408 656e7472 7920696e 20646174 61626173  entry in databas
 804b418 650a000a 6c6f6b69 643a2063 6c69656e  e...lokid: clien
 804b428 74203c25 643e2066 72656564 2066726f  t <%d> freed fro
 804b438 6d206c69 7374205b 25645d00 2f737461  m list [%d]./sta
 804b498 73747269 6e670a00 0a6c6f6b 69643a20  string...lokid:
 804b4a8 636c6965 6e74203c 25643e20 72657175  client <%d> requ
 804b4b8 65737465 64206120 70726f74 6f636f6c  ested a protocol
 804b4c8 20737761 700a0009 73656e64 696e6720   swap...sending
 804b4d8 70726f74 6f636f6c 20757064 6174653a  protocol update:
 804b4e8 203c2564 3e202573 205b2564 5d0a006c   <%d> %s [%d]..l
 804b4f8 6f6b6964 3a207472 616e7370 6f727420  okid: transport
 804b508 70726f74 6f636f6c 20636861 6e676564  protocol changed
 804b518 20746f20 25730a00 0a257300            to %s...%s.
```

The comment section is just that – clear-text information about the program. The only comment included in this binary is the name of the compiler used to compile it, or so we assume. Since this is just a comment, it can be altered with a hex editor with no effect on the programs operation, and would also change any hash value calculated from this file as a simple stealth technique.

```
Contents of section .comment:
 0000 00474343 3a202847 4e552920 322e372e  .GCC: (GNU) 2.7.
 0010 322e3100 00474343 3a202847 4e552920  2.1..GCC: (GNU)
```

The strings that are highlighted above will be parsed out into a much more readable format when the strings tool is used later in this section. For now, objdump gives us a quick look, there will be more to analyze.

From this objdump report, several items seem to further reveal the function of this binary. Each of these pieces either substantiates previous clues or adds a new clue. Fortunately, there are no items popping up that contradict the emerging picture of this binary as a host based network service provider.

Several interesting topics are mentioned in these strings, including: encryption (XOR), a client database, and changing protocol.

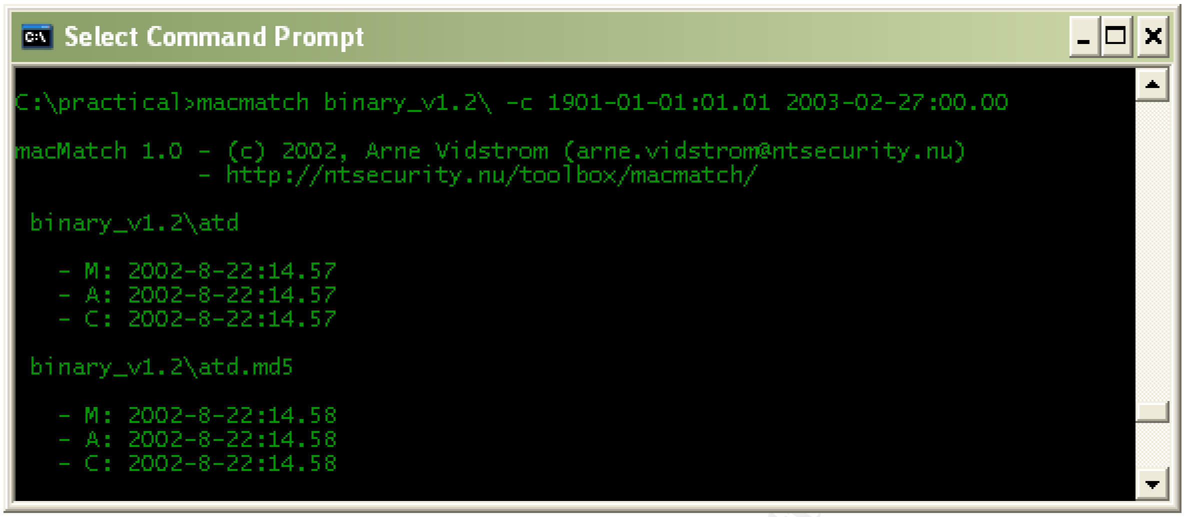
Keyword	Description
client database	A client database is either maintained and/or referenced. Strings like "client database full," "locate client entry in database," and "client [%d] freed from list" support the idea of maintaining a client database. The chdir call indicates an interest in a specific directory in the file system – possibly where this database is stored.
IP_HDRINCL socket option	A raw socket is used to send and receive raw data-grams (connection-less packets). Raw sockets do not handle the headers. This function deals with "IP HeaDeR INCLusion."
Transport protocol change	This is unusual. It refers to a condition where an apparently agreed upon protocol is changed by both the client and the server.
XOR active cryptography	This binary apparently makes use of encryption (XOR). Encryption could be applied to the client database mentioned, the data-grams, or both.

MAC Times

MAC times refer to the three date-time stamps commonly maintained for each file on many common file systems. MAC stands for **M**odified, **A**ccess, and **C**hange: when the file itself was last **m**odified, or last **a**ccessed, and when this same type of meta-data was last **c**hanged respectively. This volatile information is critical when constructing a time-line of relevant events and actions gleaned from the system being analyzed. This cements the order of events, implying the use (or not) of specific user accounts, and many other pieces of evidence that help to paint the complete picture.

Any type of file access, including seemingly benign actions like running md5sum or dir, changes at least the atime (last-accessed date-time stamp) of a file.

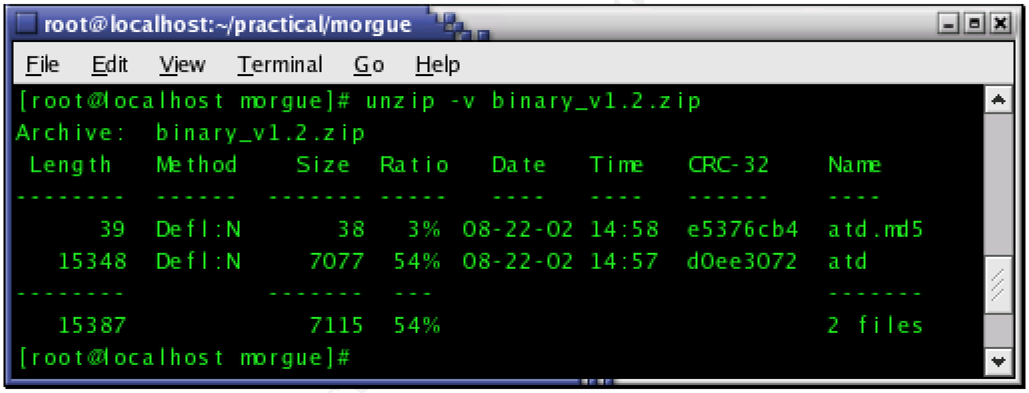
When beginning this analysis, the binary_v1.2 zip archive was unzipped and immediately listed using the dir command. This caused the atime to be changed to the current date-time, destroying the original date-time stamp. The atd files were unzipped a second time, replacing the now tainted copies with fresh copies containing their original MAC times. Arne Vidstrom's macmatch (12) tool was then used to determine the original MAC times of the subject files.



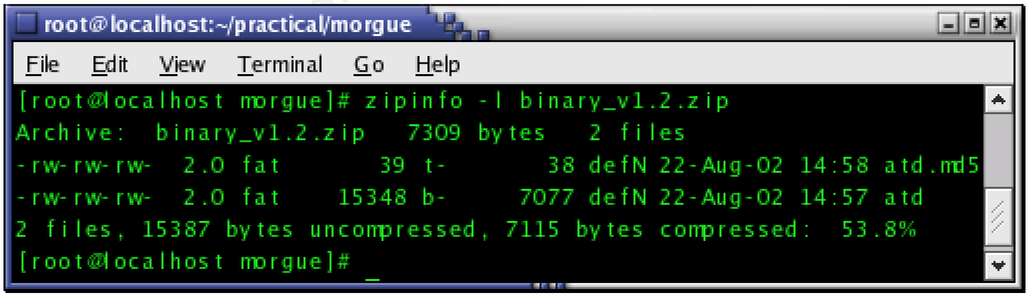
```
C:\practical>macmatch binary_v1.2\ -c 1901-01-01:01.01 2003-02-27:00.00

macMatch 1.0 - (c) 2002, Arne Vidstrom (arne.vidstrom@ntsecurity.nu)
              - http://ntsecurity.nu/toolbox/macmatch/

 binary_v1.2\atd

   - M: 2002-8-22:14.57
   - A: 2002-8-22:14.57
   - C: 2002-8-22:14.57

 binary_v1.2\atd.md5

   - M: 2002-8-22:14.58
   - A: 2002-8-22:14.58
   - C: 2002-8-22:14.58
```

Later, the Linux unzip [13] and zipinfo [14] tools confirmed these MAC times and provided additional information about the contents of the zip archive.

The -v parameter in the unzip tool provided the original date and time, as well as a CRC-32 checksum.


root@localhost:~/practical/morgue
File Edit View Terminal Go Help

```
[root@localhost morgue]# unzip -v binary_v1.2.zip
Archive:  binary_v1.2.zip
 Length   Method    Size  Ratio   Date   Time   CRC-32    Name
--------  ------  ------- -----   ----   ----   ------    ----
      39  Defl:N       38   3%  08-22-02 14:58  e5376cb4  atd.md5
   15348  Defl:N     7077  54%  08-22-02 14:57  d0ee3072  atd
--------          -------  ---                            -------
   15387             7115  54%                            2 files
[root@localhost morgue]#
```


root@localhost:~/practical/morgue
File Edit View Terminal Go Help

```
[root@localhost morgue]# zipinfo -l binary_v1.2.zip
Archive:  binary_v1.2.zip   7309 bytes   2 files
-rw-rw-rw-  2.0 fat       39 t-       38 defN 22-Aug-02 14:58 atd.md5
-rw-rw-rw-  2.0 fat    15348 b-     7077 defN 22-Aug-02 14:57 atd
2 files, 15387 bytes uncompressed, 7115 bytes compressed:  53.8%
[root@localhost morgue]# _
```

Given these MAC dates, there are two items that seem unusual:

1. Files in this zip archive have only their read and write bits set.
2. The file system on which these two files were zipped used version 2.0 of the fat file system.

These two points are unusual and unexpected.

First the atd file is an executable binary (as confirmed later in this investigation), but it is not flagged for execution. Rather than having a mask that includes the "x" attribute, atd has a 0666 mask. A 0666 (-rw-rw-rw) mask permits the file to be read and written by anyone, but it does not allow from execution. While a 0666 mode is reasonable and expected in many cases, it does not make sense for this file or this situation. One point that comes to mind is the habit of some in the hacker community to "geek" alphanumeric characters to convey an ethic to "think outside of the box", and not to be bound by "syntax rules for lesser mortals." Here, the 0666 could be intended as a "got ya" message rather than a functional file mode.

Second, this binary is an ELF binary (also confirmed later in the investigation), meaning it is a program that has been compiled specifically for the Linux platform. The odd point here is that we have a Linux program being zipped on/by a Windows system.

Conditions this atypical usually are not accidents. We can assume for the moment that the person who last worked with these files had a reason(s) for these unusual conditions, and that reason is probably relevant to the investigation.

An initial hypothesis is that the ELF binary was pulled off a Linux host onto a DOS/Windows host, where its md5 hash was calculated and written to the atd.md5 file; finally, both files were zipped into a compressed archive.

To test this hypothesis, this process was performed step-by-step, recording the MAC times at each point. A command file (lsmac.cmd) containing three dir commands, each with one of the MAC time parameters, was used to run the ls tool (ls -o --time=ctime --full-time) 3 times, recording the Modified-Change-Access time of the subject file.

The state of the cat binary was recorded using the ls, file, and stat tools. This simply documents the condition of cat before this process begins. The stat program reports a different date for each of the MAC times for the atd file. The cat binary was copied from an ext2 file system into a fresh NTFS folder via floppy diskette. Once cat is on the ntfs system, it is queried with the Windows version of ls to establish its settings. ls reports that the both the	#ls –o cat -rwxr-xr-x 1 root 14812 Aug 14 2001 cat #file cat cat: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), stripped #stat cat Size: 14812 Blocks: 32 IO Block: 4096 Regular File Device: 1642h/5698d Inode: 611859 Links: 1 Access: (0755/-rwxr-xr-x) Uid: (0/root) Gid: (0/root) Access: Wed Apr 23 10:50:20 2003 Modify: Tue Aug 14 15:47:06 2001 Change: Wed Mar 5 09:02:51 2003 C:\ copy a:\cat c:\ C:\> ls -o --time=atime --full-time a:\cat modified -rw-rw-rw- 1 user 14812 Wed Apr 23 10:50:20 2003 a:\cat

access and change dates now match the original modified date.	change -rw-rw-rw- 1 user 14812 Wed Apr 23 10:50:20 2003 a:\cat access -rw-rw-rw- 1 user 14812 Wed Apr 23 00:00:00 2003 a:\cat C:\> lsmac modified -rw-rw-rw- 1 user 14812 Wed Apr 23 10:50:20 2003 cat change -rw-rw-rw- 1 user 14812 Wed Apr 23 10:59:24 2003 cat access -rw-rw-rw- 1 user 14812 Wed Apr 23 10:59:24 2003 cat
The md5 hash was calculated for cat, and the output was redirected into the cat.md5 file. This did not alter any of the MAC dates.	C:\>md5sum cat >> cat.md5 C:\> lsmac modified -rw-rw-rw- 1 user 14812 Wed Apr 23 10:50:20 2003 cat change -rw-rw-rw- 1 user 14812 Wed Apr 23 10:59:24 2003 cat access -rw-rw-rw- 1 user 14812 Wed Apr 23 10:59:24 2003 cat C:\>cat cat.md5 \30bef954ee5b8df11101aa1d7ba531fd *C:cat

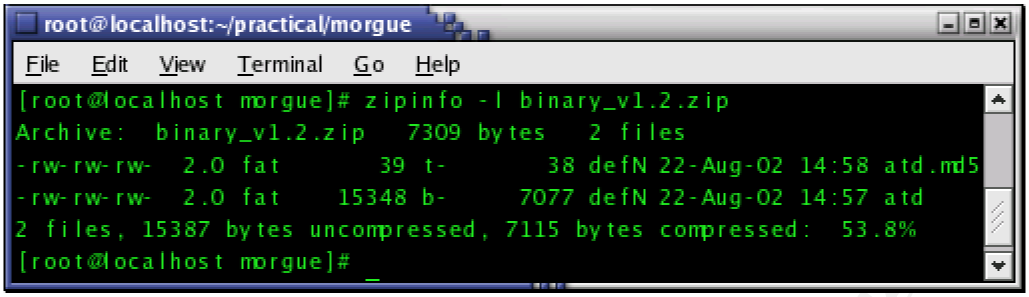
These two files were zipped into a compressed archive.	C:\>zip cat.zip cat cat.md5
	  adding: cat (92 bytes security) (deflated 52%)
	  adding: cat.md5 (92 bytes security) (stored 0%)

Next the cat file and its md5 log are deleted from this folder.
 	
This way, only the copy that has been zipped will exist in subsequent steps.
	C:\>del cat
	
	C:\>del cat.md5

The unzip program is used with the long format (-l) and verbose (-v) parameters to list the contents of the archive file. Of interest are the date and time stamps.	C:\practical>unzip -l -v cat.zip
	Archive: cat.zip
	 Length Method Size Ratio Date Time CRC-32 Name
	-------- ------ ------- ----- ---- ---- ------ ----
	   14812 Defl:N 7038 53% 04-23-03 10:50 60e1f3de cat
	      55 Stored 55 0% 04-23-03 11:00 3c4ed399 cat.md5
	-------- ------- --- -------
	   14867 7093 52% 2 files

The files are unzipped, and the MAC time is rechecked. The change and modified time continues to remain unchanged while the access date has been updated.	C:\practical>unzip cat.zip Archive: cat.zip inflating: cat extracting: cat.md5 C:\> unzip cat.zip modified -rw-rw-rw- 1 user 14812 Wed Apr 23 10:50:20 2003 cat change -rw-rw-rw- 1 user 14812 Wed Apr 23 10:59:24 2003 cat access -rw-rw-rw- 1 user 14812 Wed Apr 23 11:01:27 2003 cat

This exercise shows that only the access time is altered during this hypothetical scenario. However, it does not account for the MAC times seen in the atd files or the change in the permissions.


root@localhost:~/practical/morgue
File Edit View Terminal Go Help

```
[root@localhost morgue]# zipinfo -l binary_v1.2.zip
Archive:  binary_v1.2.zip   7309 bytes   2 files
-rw-rw-rw-  2.0 fat       39 t-        38 defN 22-Aug-02 14:58 atd.md5
-rw-rw-rw-  2.0 fat    15348 b-      7077 defN 22-Aug-02 14:57 atd
2 files, 15387 bytes uncompressed, 7115 bytes compressed:  53.8%
[root@localhost morgue]# _
```

Once the binary_v1.2.zip was unzipped, the stat tool was used to capture MAC and other data. While the change time had been set to the current date-time, the Access and Modify timestamps are the same as reported by other tools. This does not say anything about the validity of those timestamps, only that they are consistently reported.

```
# stat MACPreservationZone/*

  File: "MACPreservationZone/atd"
  Size: 15348           Blocks: 32        IO Block: 4096   Regular File
Device: 303h/771d       Inode: 1864870    Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Aug 22 14:57:54 2002
Modify: Thu Aug 22 14:57:54 2002
Change: Tue Apr 22 20:53:31 2003

  File: "MACPreservationZone/atd.md5"
  Size: 39          Blocks: 8         IO Block: 4096   Regular File
Device: 303h/771d       Inode: 1864867    Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Aug 22 14:58:08 2002
Modify: Thu Aug 22 14:58:08 2002
Change: Tue Apr 22 20:53:31 2003
```

This does show that the hypotheses is supported, that the MAC dates on the suspect files could have been produced in this way. But it does not explain why all three MAC dates are identical, 2002-8-22:14.55.

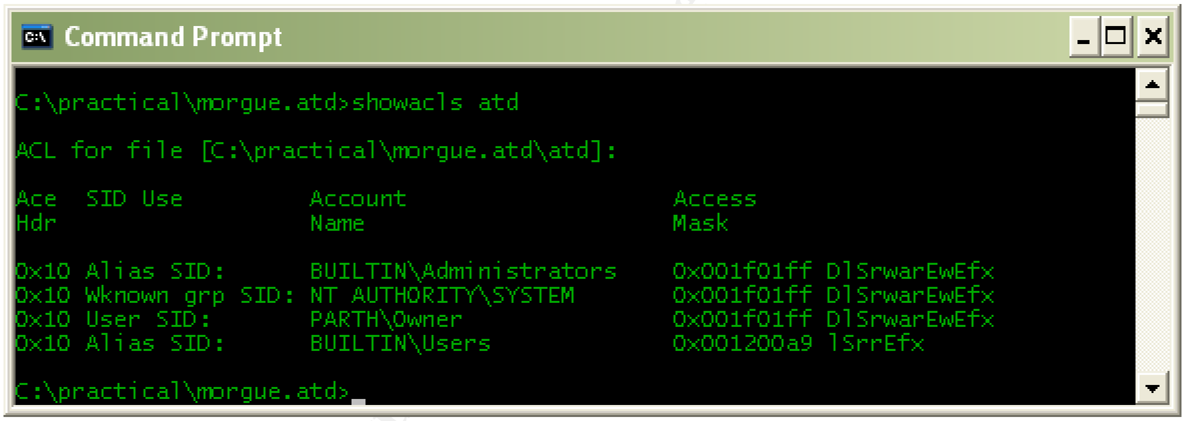
So, what do these MAC times tell us? At this point, very little. Manipulating a MAC time to any date-time value an intruder wants is trivial. However, the analysis will establish a degree of trust in this information based on conditions such as the coloration between these dates and trusted historic information found on the Internet, along with internal references to dateable information.

One obvious explanation for the three MAC times being identical is that the atd file has not been accessed since it was create. Another is that the file had been moved from one file system to another in such a combination that the original three MAC times were replaced during the copy process.

Since neither of these two possibilities seem likely, a third possibility should be considered; that the MAC times have been intentionally altered, which would indicate malicious manipulation and intentionally altering evidence.

File owner(s) – (user and/or group)

The showacls.exe [15] program from the NT Resource Kit reported the current owner/group information rather than the original. The help screen (showacls.exe /?) explained the Access Mask values, which, in this case, were of no real interest. However, if the initial indications are correct and this is an ELF binary that was archived (zipped) on a win32 system, user and group information will probably have been inadvertently stripped along the way. Metadata such as "file owner" is usually stored as an ID number that is matched to a local user account table. In Linux, the root account is 0; while in Windows, the administrator account is 500. This, along with other metadata field incompatibilities, would explain the loss of such metadata when moving a file between these operating systems. So, it is not likely that the original owner, or similar metadata, can be recovered from this copy of the binary.



```
C:\practical\morgue.atd>showacls atd

ACL for file [C:\practical\morgue.atd\atd]:

Ace  SID Use          Account                    Access
Hdr                   Name                       Mask

0x10 Alias SID:       BUILTIN\Administrators     0x001f01ff DlSrwarEwEfx
0x10 Wknown grp SID:  NT AUTHORITY\SYSTEM        0x001f01ff DlSrwarEwEfx
0x10 User SID:        PARTH\Owner                0x001f01ff DlSrwarEwEfx
0x10 Alias SID:       BUILTIN\Users              0x001200a9 lSrrEfx

C:\practical\morgue.atd>
```

Below are selected portions of the zipinfo.rpt report file. Notice that atd was zipped on a Microsoft system. Zipinfo also reports that this zip archive is not encrypted, which saves us from having to crack or discover its password.

Then again, the modified time is confirmed, and we are told there are no embedded comments in this zip archive file. These comment areas are of interest because attackers have been known to include passwords and useful information in comment areas.

```
# zipinfo -ltv binary_v.1.2.zip
Archive:  binary_v1.2.zip   7309 bytes   2 files
  atd
  file system or operating system of origin:       MS-DOS, OS/2 or NT
FAT
  version of encoding software:                    2.0
  file security status:                            not encrypted
  file last modified on (DOS date/time):           2002 Aug 22
14:57:54
  apparent file type:                              binary
```

There is no **file comment.**

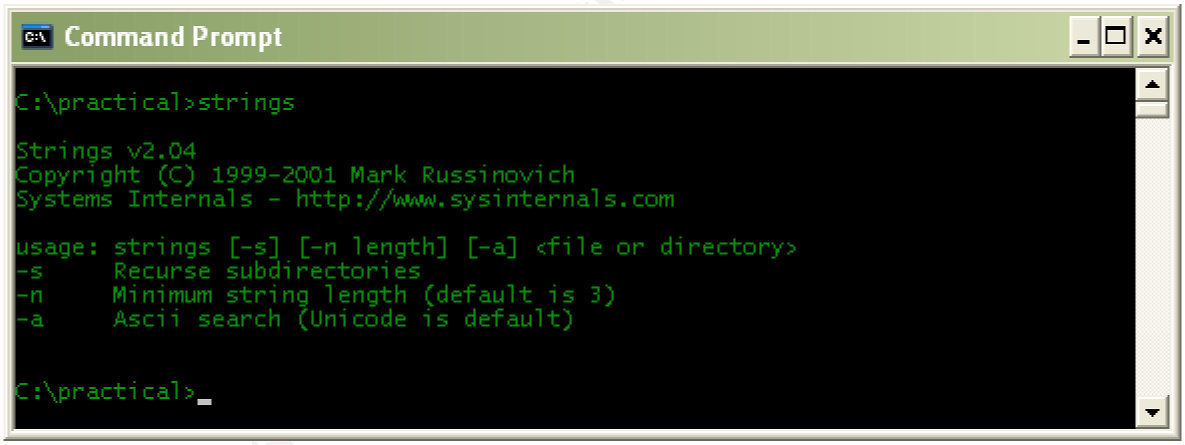
Keywords Found That Are Associated With The Program/File

Using the Win32 strings [19] tool from System Internals, both the Unicode and ASCII strings within the atd file were captured.

By default, this version of strings searches for Unicode strings rather than ASCII. Since this is a Windows tool, and Windows is predominantly and increasingly using Unicode, an analyst should expect to encounter Unicode more often than ASCII. This assumes that strings is being run against binary files (part of or produced by the operating system) since non-binary files are more likely intended for human viewing and are consequently ASCII encoded.

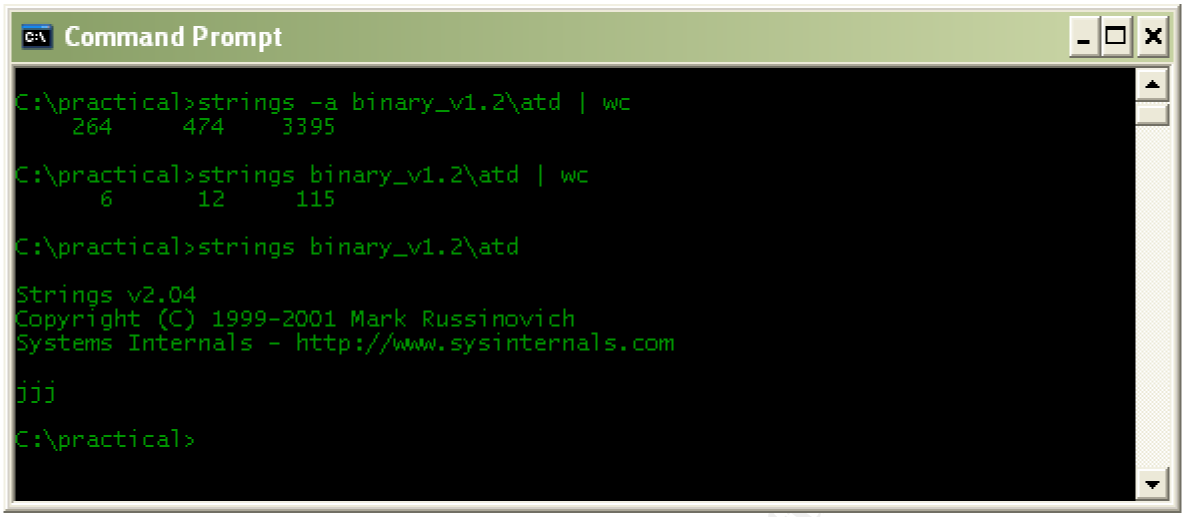
When you consider that this is an ELF binary from a Windows file system, it gets even more convoluted. Which encoding should be expected, ASCII or Unicode?

This reinforces the fact that this binary cannot be trusted, especially to conform to my preconceived expectations that only ASCII or Unicode will exist in this ELF binary. After all, anyone who could code a program like this, could easily embed material encoded however they would like. Actually, embedding or injecting Unicode text in an ASCII environment might satisfy such an attacker's sense of humor, or ego.



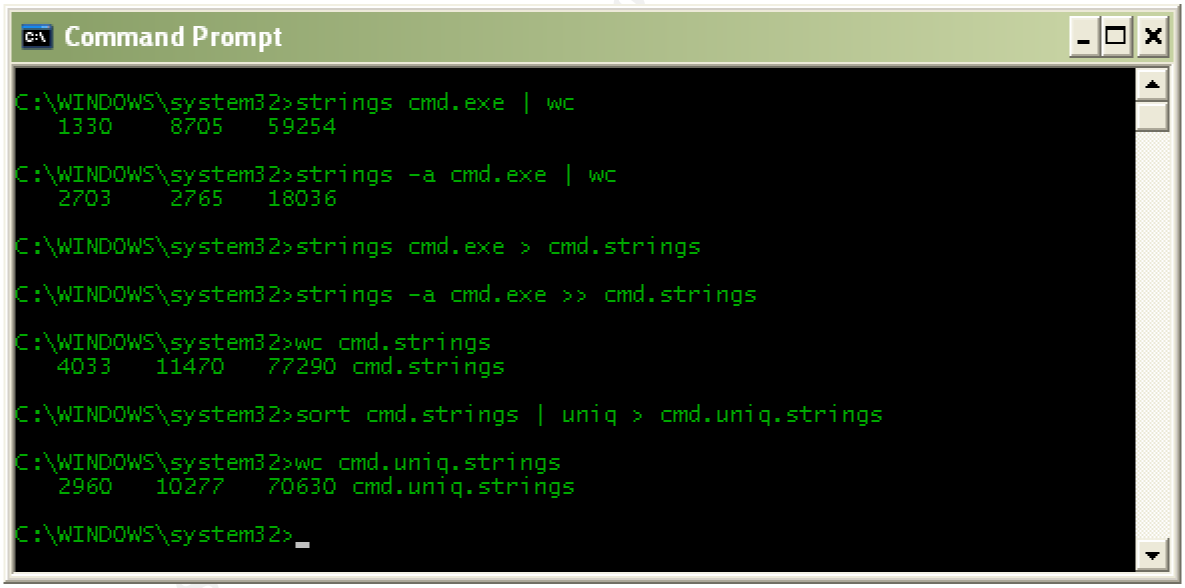
```
C:\practical>strings

Strings v2.04
Copyright (C) 1999-2001 Mark Russinovich
Systems Internals - http://www.sysinternals.com

usage: strings [-s] [-n length] [-a] <file or directory>
-s     Recurse subdirectories
-n     Minimum string length (default is 3)
-a     Ascii search (Unicode is default)


C:\practical>_
```

I first compared the difference in results between Unicode and ASCII. The string "jjj" was the only Unicode string identified (6 times). The assumption was made that this was an anomaly, and all the actual strings were in ASCII format.



```
C:\practical>strings -a binary_v1.2\atd | wc
    264     474    3395

C:\practical>strings binary_v1.2\atd | wc
      6      12     115

C:\practical>strings binary_v1.2\atd

Strings v2.04
Copyright (C) 1999-2001 Mark Russinovich
Systems Internals - http://www.sysinternals.com

jjj

C:\practical>
```

While Windows, beginning with NT 4.0, makes extensive use of Unicode, a Windows executable binary often contains both ASCII and Unicode strings of interest to an investigator. In the case of Win32 binary files, it is helpful to parse the ASCII and Unicode strings and then combine and organize them (sort [5] and uniq [5]) them for manual review. This process is illustrated and described below.



```
C:\WINDOWS\system32>strings cmd.exe | wc
   1330    8705   59254

C:\WINDOWS\system32>strings -a cmd.exe | wc
   2703    2765   18036

C:\WINDOWS\system32>strings cmd.exe > cmd.strings

C:\WINDOWS\system32>strings -a cmd.exe >> cmd.strings

C:\WINDOWS\system32>wc cmd.strings
   4033   11470   77290 cmd.strings

C:\WINDOWS\system32>sort cmd.strings | uniq > cmd.uniq.strings

C:\WINDOWS\system32>wc cmd.uniq.strings
   2960   10277   70630 cmd.uniq.strings

C:\WINDOWS\system32>_
```

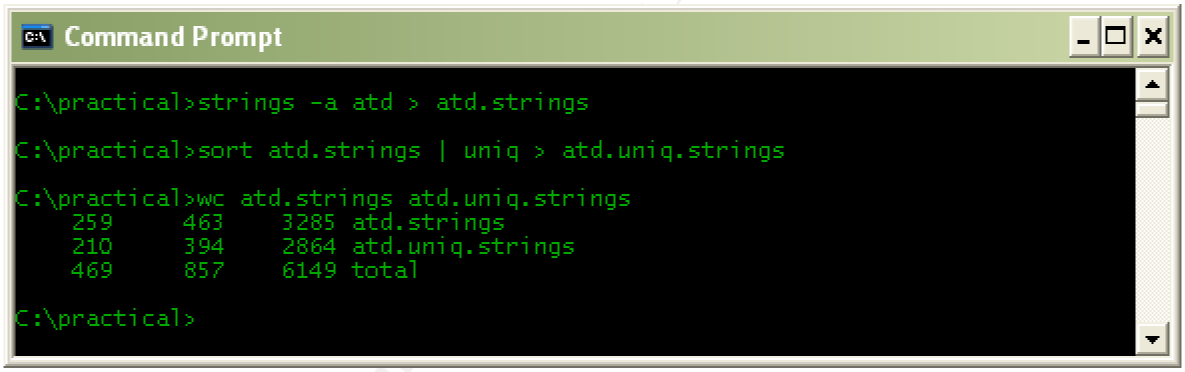
At this point I had the “raw” strings ready for review. However, to make the initial review of the strings easier and more likely fruitful, I first sorted and de-duped (removed duplicate items) the strings’ results. This helped to quickly identify strings of interest for more focused investigation.

The strings tool pre-appends its own identification information to the output. So, before sorting and de-duping these files, that identification material was manually removed with an editor, leaving only the strings parsed from the binaries.

The wc [5] (word count) program reported significant distilling of the strings down to their core items – reducing the background noise during the initial manual review of the strings. In this case, 50 strings, or 19% reduced the number of strings found in atd.

Given the huge amount of raw data that can be produced by feeding a subject file into a battery of analysis tools, it is important to reduce the volume of data to a manageable size while not losing important clues. A specific string may provide the best clue relevant to the investigation, but it may also be the frequency of an otherwise benign string that provides that clue. For example, the presence of a "noop" command (no operation) may not be suspicious on its own, but to find dozens of "noop" strung together could indicate a "noop-slide", which is an indicator of a buffer overflow exploit.

> NOP is 'No Operation' CPU command. It does precisely nothing. *NOP slide (also called NOOP sled) is used in buffer overflow attacks to compensate for imprecise knowledge of offset to jump at. Having a string of NOPs in front the actual attack shell code in the buffer allows one to jump somewhere in the middle of the NOP slide and still have the shell code executed.* [36]



```
C:\practical>strings -a atd > atd.strings

C:\practical>sort atd.strings | uniq > atd.uniq.strings

C:\practical>wc atd.strings atd.uniq.strings
    259     463    3285 atd.strings
    210     394    2864 atd.uniq.strings
    469     857    6149 total

C:\practical>
```

Of these remaining 210 strings, a few jump out as helpful in identifying the details of this program.

Strings like those below indicate this program uses a network connection.

Keyword	Description
active transport:	This refers to the transport layer in the TCPIP and OSI network models, and has to do with the mechanics of operating a logical session between two host applications.
server uptime:	This appears to be a report field label, that shows the time this or another server have been active.
sending protocol update: <%d> %s [%d]	This appears to be a format strings. The literal text "sending protocol update:" is followed by the formatted (for display) values within specific variables.

These strings seem to indicate the display of server status and operation information. That would make this more likely to be a daemon [37] rather than a client application.

Other strings indicate that this file was compiled using the GNU gcc 2.7.2.1 compiler. This will be helpful when attempting to compile a fresh copy of this program. This version was included in Redhat Linux 4.2 [38]

Keyword	Description
GCC: (GNU) 2.7.2.1	This is the GNU (*GNU is a recursive acronym for ``GNU's Not Unix''; it is pronounced "guh-NEW"*) gcc (*GNU Compiler Collection*) version 2.7.2.1 compiler for the C language. According to the gnu page, this version was released on June 29, 1996 [38]
/lib/ld-linux.so.1	Summary: A dynamic loader for a.out format files. The ld.so package contains the shared library configuration tool, ldconfig, which is required by many packages. It also includes the shared library loader and dynamic loader for Linux libc 5. This version was released on Feb 3, 2000 [39]
Libc.so.5	Contains the standard libraries that are used by multiple programs on the system. In order to save disk space and memory, as well as to ease upgrades, common system code is kept in one place and shared between programs. This package contains the most important sets of shared libraries, the standard C library and the standard math library. Without these, a Linux system will not function. This software was released on Oct 8, 1997 [40]

Given the dates of the above software; while the c compiler is notably old, the most recent date (Feb 3, 2000) represents the earliest this binary could have been compiled. So, the binary was compiled between Feb 3, 2000 and Aug 22, 2002 (the files MAC times). This will make recreating the binary difficult because the identical library and compiler files, along with the compiler options must be identical to the original if there is any chance to recreate an identical copy of the binary.

Also found were strings that indicate the use of, or options for, encryption.

Keyword	Description
.hash	A hash is the process or algorithm that accepts variable length input and produces a fixed length value that is unique to the input material. However, the original material cannot be reproduced from the hash value. Hashes are often used to fingerprint digital material so that at a later time the hash can be recalculated and compared to the original in order to verify that both copies of the material are identical at the bit level.
Active cryptography:	This is presumably a title used to report the cryptographic mechanism currently in use.
XOR	XOR is a very old and established cryptographic algorithm. XOR is well understood, fast, and though not the strongest, it is strong enough for many purposes.

Finally, several strings were noted that would help identify the program itself. "lokid" is mentioned several times, apparently as the name of the program. For example, the first line appears to illustrate the program's acceptable syntax. After that, the label for the program's version, a canned error message, and finally a copyright notice all support the hypothesis that the program's name is "lokid." Other strings imply that this is a daemon, which is supported by the trailing "d" in the name "loki**d**".

Keyword	Possible Description
LOKI2	Possibly the name of this program.
lokid -p (i|u) [-v (0|1)]	Lokid appears to be the program's name; with the ending "d" implying this is a daemon service. The entire string looks like an example of the programs accepted syntax.
lokid version:	This might be a heading used when displaying the program's version.
lokid: Client database full	This looks like an error or status message either informing a client why they are not being granted access, or as a descriptive entry for an administrative console or log.
lokid: clean exit (killed at client request)	This looks like a status message for the administrator or system log.
route [(c) 1997 guild corporation worldwide]	This is a copyright notice.

This is plenty of data to begin searching the Internet for detailed information about a *network Linux daemon* named *lokid*.

Searching Google.com with the following keywords produced several useful hits.

1. LOKI2 linux – 222
2. LOKI2 Phrack - 192
3. LOKI2 linux security - 143
4. network Linux ICMP lokid – 72
5. Linux daemon lokid – 65
6. network Linux ICMP lokid "guild corporation" – 43

Not only did these sources provide detailed background information about the Loki2 program components, including the lokid daemon and client programs, but also included the source code.

Program Description

What type of program is this?

By the end of this section it will be shown that this atd file is actually the lokid.c file; the server portion of a client-server covert communications suite that takes advantage of an exploitable characteristic of the ICMP (Internet Connection Management Protocol) protocol.

Loki2 takes advantage of a condition in ICMP, which also happens to exist in many other protocols in the TCP/IP family of network protocols. RFC (Request For Comment) 792 [(23)] defines ICMP. Loki specifically takes advantage of the fact that ICMP allows for arbitrary binary material to be injected into each ICMP packets without altering their operation.

The ICMP RFC defines 15 types of packets. Of those, two are used by Loki2 to create a covert communications channel between two hosts: one running as a server and the other as a client. Normally, a type 08 ICMP packet requests a response (in the form of an ICMP type 00 packet) from a remote host. This is the ping process common to TCP/IP networks.

A ping is used to verify that a remote host is currently active, meaning its TCP/IP stack is functioning and the host has connectivity to the network. Network support staff also use pings to verify the network path to another host.

A Loki2 client used ICMP request response (type 08) packets to send commands or other material to a loki2d server. The loki2d server (which atd appears to be) uses ICMP request response (type 00) packets to return the results of the client's commands.

What is this program used for?

From the perspective of a device observing this exchange, one host is simply pinging another, and it replies.

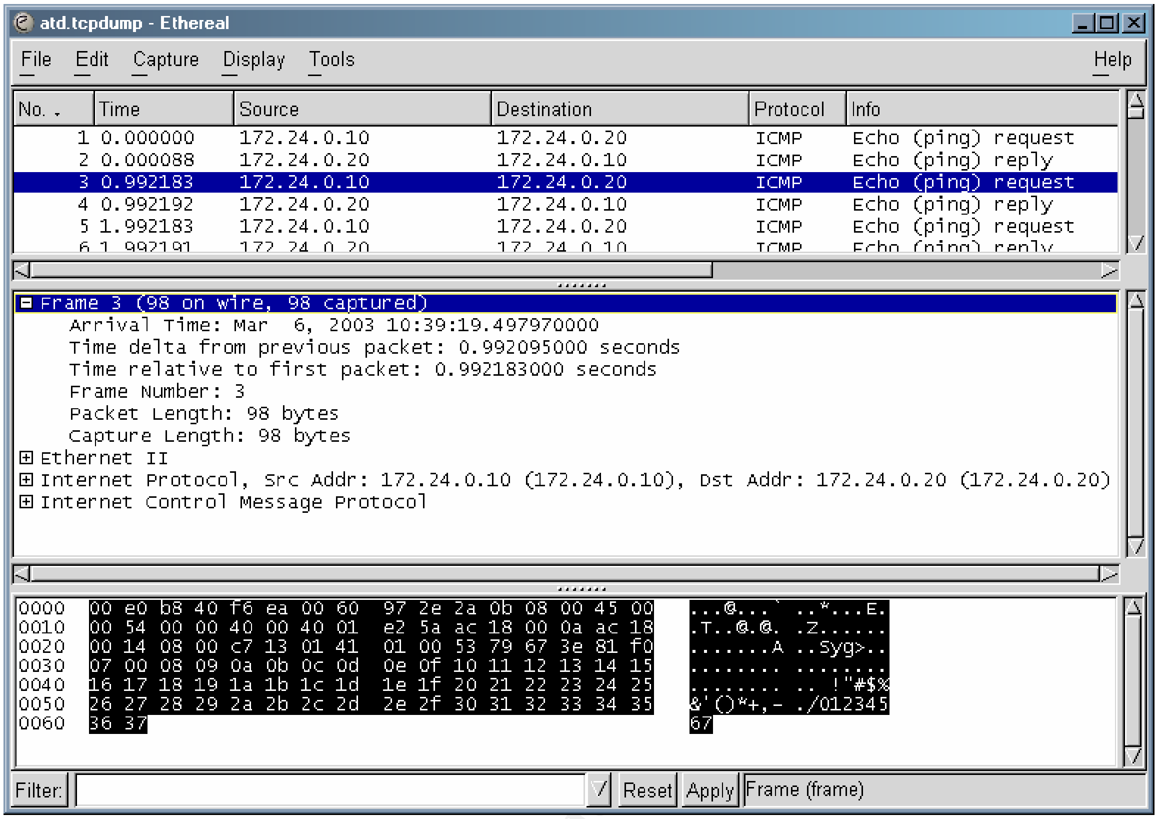
The covert characteristic exists because ICMP packets are assumed by network routing and filtering devices (routers, firewalls, IDS agents) to transport only specific protocol management information. Consequently, ICMP packets of all types are commonly allowed to traverse public and private networks unhindered and not inspected. There exists in each ICMP packet an undefined area in which any type of payload can be stored and transmitted. For the sake of speed and efficiency, most network devices choose to ignore this type of undefined field and interrogate only the control fields.

To better understand how a covert ICMP channel would look, the ethereal [51] tool does a great job of dissecting network packets and putting them into "conversational context" for detailed examination. Below, a single ping echo request followed by the ping echo reply illustrates a covert ICMP channel. The channel characteristic is seen when large numbers of these request/reply pairs are used to transmit a digital conversation, file transfer, or session of some type.

To start the process, 172.24.0.10 issues an ICMP ECHO request (type 08) addressed to 172.24.0.20.

In the illustration below, you can see a summary panel at the top and a breakdown of the packet components in the middle panel. The protocol encapsulation is clearly defined with the lower level of the network protocol stack listed first, then moving on the deeper Ethernet, IP, and finally the ICMP protocol.

As you highlight a packet layers in the middle pane, that portions of the packet is display in the bottom pane and is highlighted to set it apart from other protocol components. In this illustration, the entire 98 bytes of this packet is highlighted in the bottom panel.

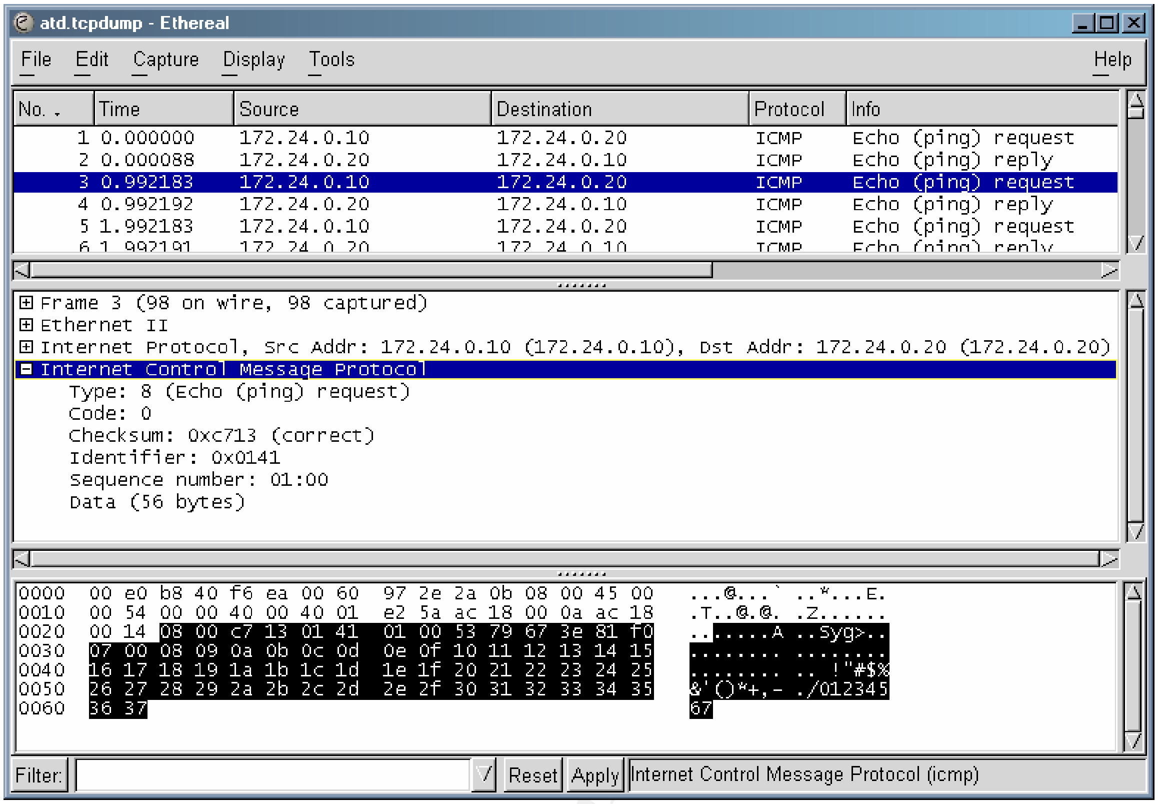


```
No. .  Time      Source       Destination  Protocol Info
     1 0.000000  172.24.0.10  172.24.0.20  ICMP     Echo (ping) request
     2 0.000088  172.24.0.20  172.24.0.10  ICMP     Echo (ping) reply
     3 0.992183  172.24.0.10  172.24.0.20  ICMP     Echo (ping) request
     4 0.992192  172.24.0.20  172.24.0.10  ICMP     Echo (ping) reply
     5 1.992183  172.24.0.10  172.24.0.20  ICMP     Echo (ping) request
     6 1.992191  172.24.0.20  172.24.0.10  ICMP     Echo (ping) reply
```

```
⊟ Frame 3 (98 on wire, 98 captured)
      Arrival Time: Mar  6, 2003 10:39:19.497970000
      Time delta from previous packet: 0.992095000 seconds
      Time relative to first packet: 0.992183000 seconds
      Frame Number: 3
      Packet Length: 98 bytes
      Capture Length: 98 bytes
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 172.24.0.10 (172.24.0.10), Dst Addr: 172.24.0.20 (172.24.0.20)
⊞ Internet Control Message Protocol
```

```
0000   00 e0 b8 40 f6 ea 00 60  97 2e 2a 0b 08 00 45 00    ...@...` ..*...E.
0010   00 54 00 00 40 00 40 01  e2 5a ac 18 00 0a ac 18    .T..@.@. .Z......
0020   00 14 08 00 c7 13 01 41  01 00 53 79 67 3e 81 f0    .......A ..Syg>..
0030   07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15    ........ ........
0040   16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25    ........ .. !"#$%
0050   26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35    &'()*+,- ./012345
0060   36 37                                               67
```

A closer look at the ICMP portion of this same packet shows that the encapsulated ICMP portion of the packet is indeed an ICMP type 08 echo request packet. Notice the \x08 byte, the first byte that is highlighted, on line 0020.



```
No.  Time      Source       Destination  Protocol  Info
  1 0.000000  172.24.0.10  172.24.0.20  ICMP      Echo (ping) request
  2 0.000088  172.24.0.20  172.24.0.10  ICMP      Echo (ping) reply
  3 0.992183  172.24.0.10  172.24.0.20  ICMP      Echo (ping) request
  4 0.992192  172.24.0.20  172.24.0.10  ICMP      Echo (ping) reply
  5 1.992183  172.24.0.10  172.24.0.20  ICMP      Echo (ping) request
  6 1.992191  172.24.0.20  172.24.0.10  ICMP      Echo (ping) reply
```

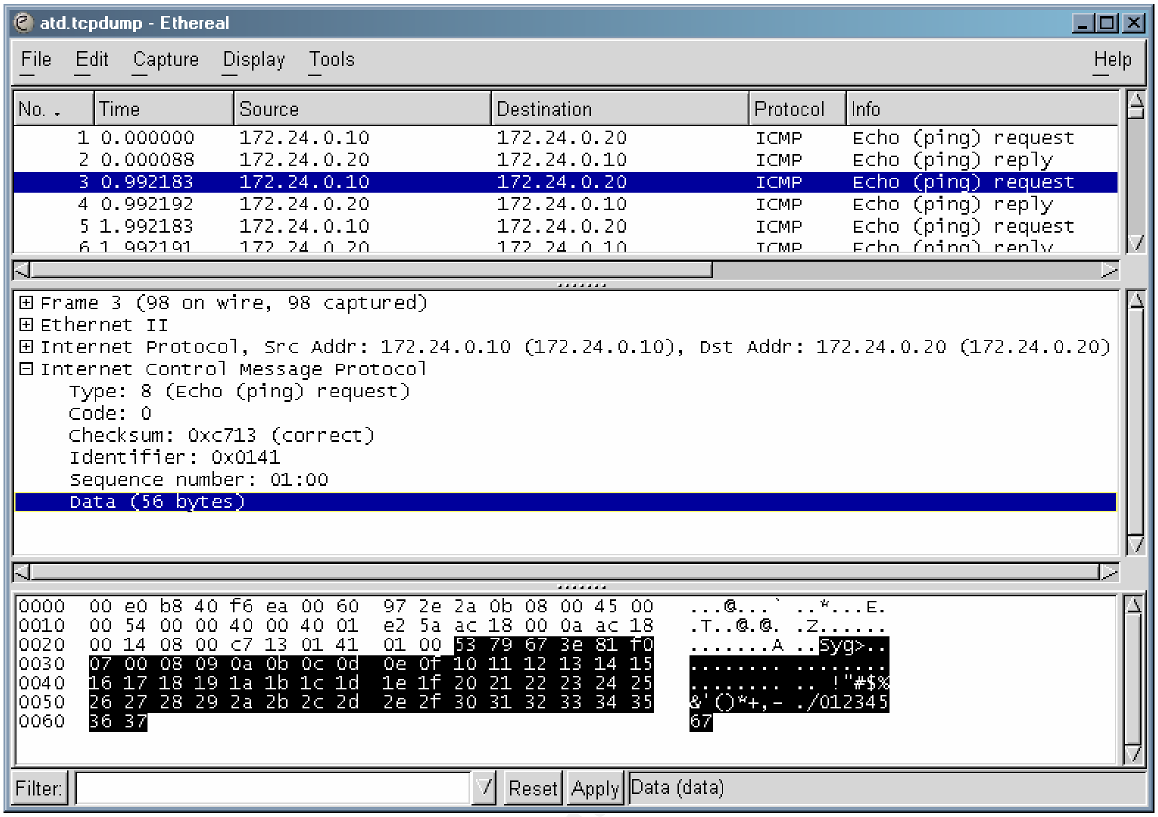
```
⊞ Frame 3 (98 on wire, 98 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 172.24.0.10 (172.24.0.10), Dst Addr: 172.24.0.20 (172.24.0.20)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xc713 (correct)
    Identifier: 0x0141
    Sequence number: 01:00
    Data (56 bytes)
```

```
0000  00 e0 b8 40 f6 ea 00 60  97 2e 2a 0b 08 00 45 00   ...@...` ..*...E.
0010  00 54 00 00 40 00 40 01  e2 5a ac 18 00 0a ac 18   .T..@.@. .Z......
0020  00 14 08 00 c7 13 01 41  01 00 53 79 67 3e 81 f0   .......A ..Syg>..
0030  07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

Internet Control Message Protocol (icmp)

A few bytes worth of control information later brings us to an optional data area that it has no defined use, though it was apparently assumed vendors would use it for proprietary metrics or other enhancements.

Remember, the entire packet length is only 98 bytes, and here 56 of those 98 bytes, or 57%, are in the optional data area. In effect, an ICMP packet addresses and supports its intended function with only its first 42 bytes, but then hauls another 56 bytes along that have no effect on the proper function of the packet. This 56-byte area could be filled with anything and the packet would continue to function as usual. In this example, beginning at the 3rd byte on line 0030, \x08, and the remainder of the packet is simply the hex values counting from \x08 to \x37. Simple filler.



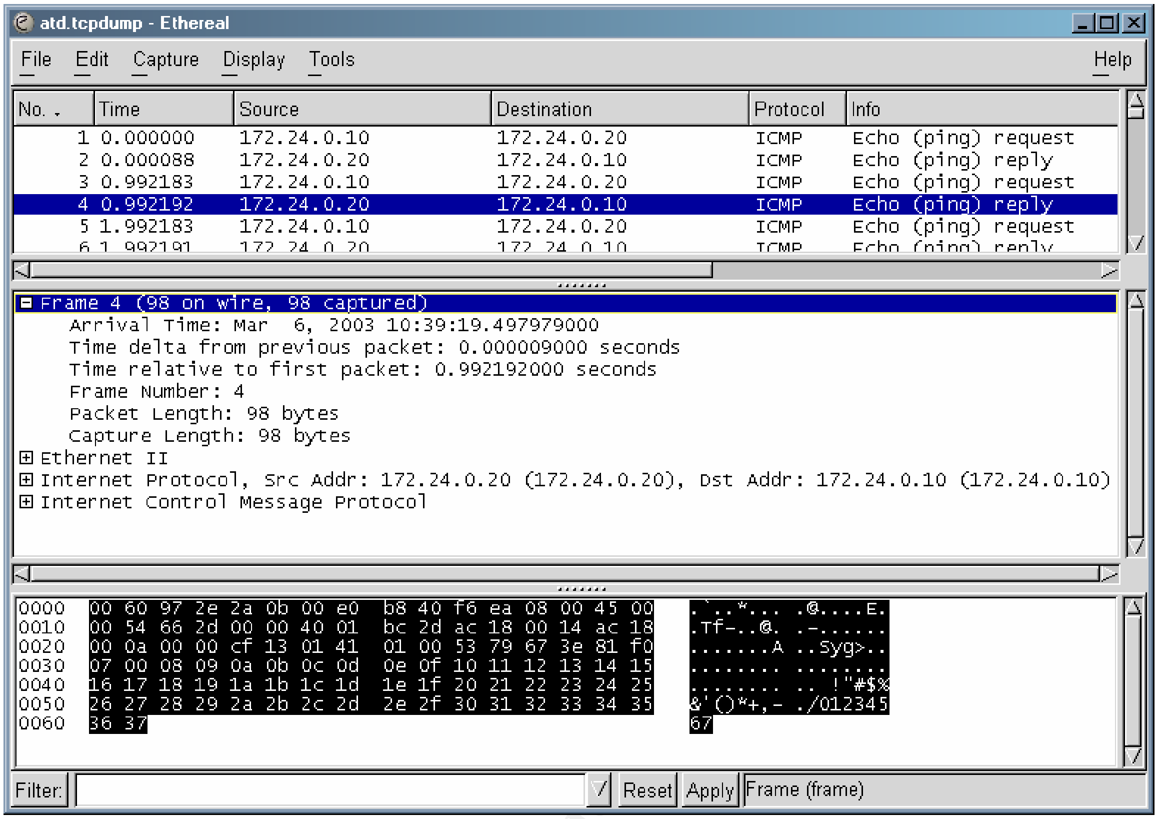
```
No.  Time        Source         Destination    Protocol  Info
  1 0.000000    172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
  2 0.000088    172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
  3 0.992183    172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
  4 0.992192    172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
  5 1.992183    172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
  6 1.992191    172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
```

```
⊞ Frame 3 (98 on wire, 98 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 172.24.0.10 (172.24.0.10), Dst Addr: 172.24.0.20 (172.24.0.20)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xc713 (correct)
    Identifier: 0x0141
    Sequence number: 01:00
    Data (56 bytes)
```

```
0000  00 e0 b8 40 f6 ea 00 60  97 2e 2a 0b 08 00 45 00   ...@...` ..*...E.
0010  00 54 00 00 40 00 40 01  e2 5a ac 18 00 0a ac 18   .T..@.@. .Z......
0020  00 14 08 00 c7 13 01 41  01 00 53 79 67 3e 81 f0   .......A ..Syg>..
0030  07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

If the recipient host chooses to reply to this ICMP echo request, it replies with an ICMP echo response type 00 packet to inform the initiating host that yes, this host is on the network and functional.

As with the ICMP request packet, the ICMP reply packet is 98 bytes long. The third byte on line 0020 indicates this is an ICMP echo reply, type \x00 where this field contained \x08 in the request packet to identify it.

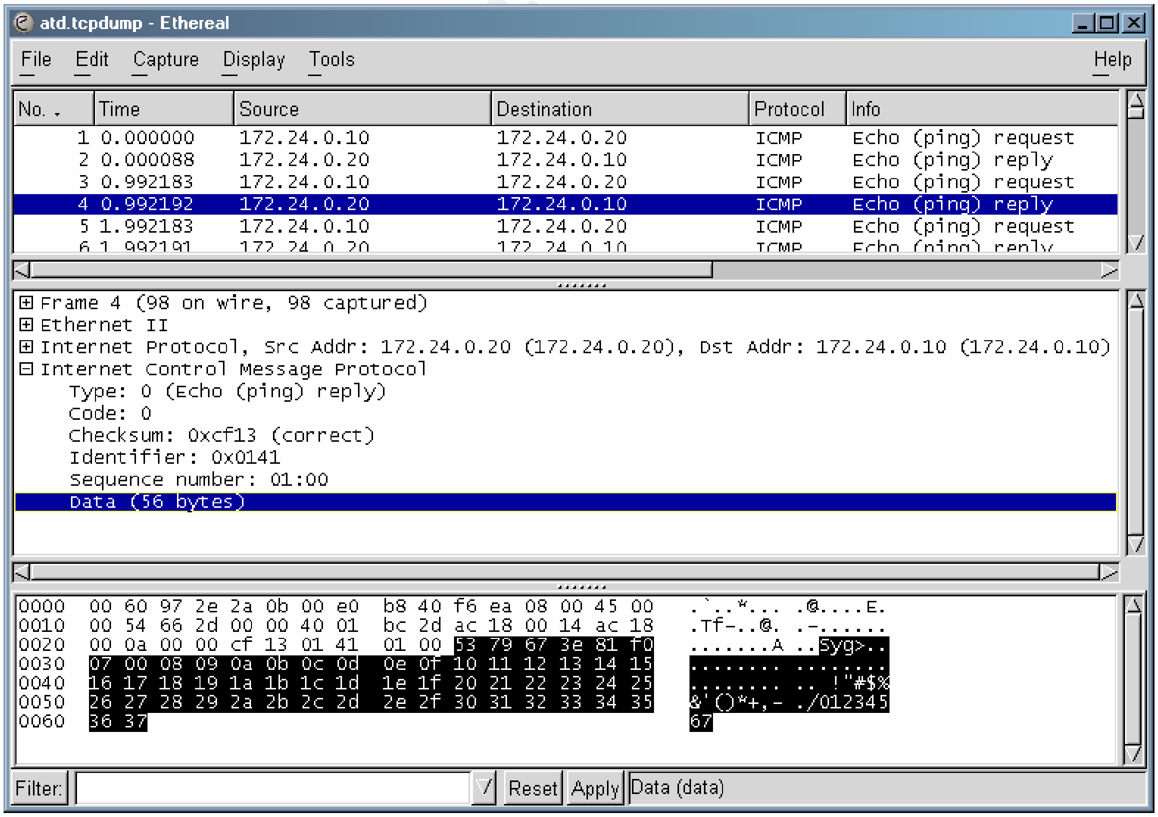


```
No.   Time       Source         Destination    Protocol  Info
    1 0.000000   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    2 0.000088   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
    3 0.992183   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    4 0.992192   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
    5 1.992183   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    6 1.992191   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
```

```
⊟ Frame 4 (98 on wire, 98 captured)
    Arrival Time: Mar  6, 2003 10:39:19.497979000
    Time delta from previous packet: 0.000009000 seconds
    Time relative to first packet: 0.992192000 seconds
    Frame Number: 4
    Packet Length: 98 bytes
    Capture Length: 98 bytes
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 172.24.0.20 (172.24.0.20), Dst Addr: 172.24.0.10 (172.24.0.10)
⊞ Internet Control Message Protocol
```

```
0000  00 60 97 2e 2a 0b 00 e0  b8 40 f6 ea 08 00 45 00   .`..*... .@....E.
0010  00 54 66 2d 00 00 40 01  bc 2d ac 18 00 14 ac 18   .Tf-..@. .-......
0020  00 0a 00 00 cf 13 01 41  01 00 53 79 67 3e 81 f0   .......A ..Syg>..
0030  07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

An ICMP request packet supports the same 56-byte data space.



```
No.   Time       Source         Destination    Protocol  Info
    1 0.000000   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    2 0.000088   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
    3 0.992183   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    4 0.992192   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
    5 1.992183   172.24.0.10    172.24.0.20    ICMP      Echo (ping) request
    6 1.992191   172.24.0.20    172.24.0.10    ICMP      Echo (ping) reply
```

```
⊞ Frame 4 (98 on wire, 98 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 172.24.0.20 (172.24.0.20), Dst Addr: 172.24.0.10 (172.24.0.10)
⊟ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xcf13 (correct)
    Identifier: 0x0141
    Sequence number: 01:00
    Data (56 bytes)
```

```
0000  00 60 97 2e 2a 0b 00 e0  b8 40 f6 ea 08 00 45 00   .`..*... .@....E.
0010  00 54 66 2d 00 00 40 01  bc 2d ac 18 00 14 ac 18   .Tf-..@. .-......
0020  00 0a 00 00 cf 13 01 41  01 00 53 79 67 3e 81 f0   .......A ..Syg>..
0030  07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

Again, the material in the data space is simply the numeric sequence from \x08 to \x37. Instead of having some functional purpose, it is simply filler.

Rather than thinking of an ICMP echo request/reply pair of packets as performing the ping process, think of them as a communications channel, composed of packets that each have a 46-byte control section, whose only purpose is to carry its 56-byte payload between two hosts. This is how the Loki program creates and sustains a covert communications channel. Not only is this covert session created and maintained, the payload can optionally be encrypted so that each packet continues what would appear to be binary white noise.

When was the last time it was used?

Another question is when was this file last executed? The stat and macmatch utilities previously reported that all three of the MAC times associated with the atd file were set to “Thur Aug 22 14:57:54 2002.” An executable program file with the same date in all three MAC times would seem to indicate that the file had not been accessed since it was created/compiled.

This makes the assumption that when a binary is compiled from source code the MAC times are all the same; later, when the executable is run, only the atime is updated. To test this assumption, a program was compiled and executed while observing the MAC times at each step.

This is similar to the process earlier that addressed the question of the MAC times associated with the atd file. Here, however, the question is about when the program was last executed, and relies on the MAC time information gathered earlier.

The mac-robber.c file was copied into a new directory, and its MAC times were recorded using the stat tool.

The file was then compiled, and the MAC times of the resulting binary were immediately recorded. Notice that all three MAC times are identical – the date and time the binary was compiled.

```
# stat mac-robber.c
Access: Fri Apr  4 20:46:36 2003
Modify: Mon Jan 21 12:48:27 2002
Change: Sat Mar  1 17:33:44 2003

# gcc mac-robber.c –o mac-robber

# stat mac-robber
Access: Sun Apr  6 22:52:41 2003
Modify: Sun Apr  6 22:52:41 2003
Change: Sun Apr  6 22:52:41 2003
```

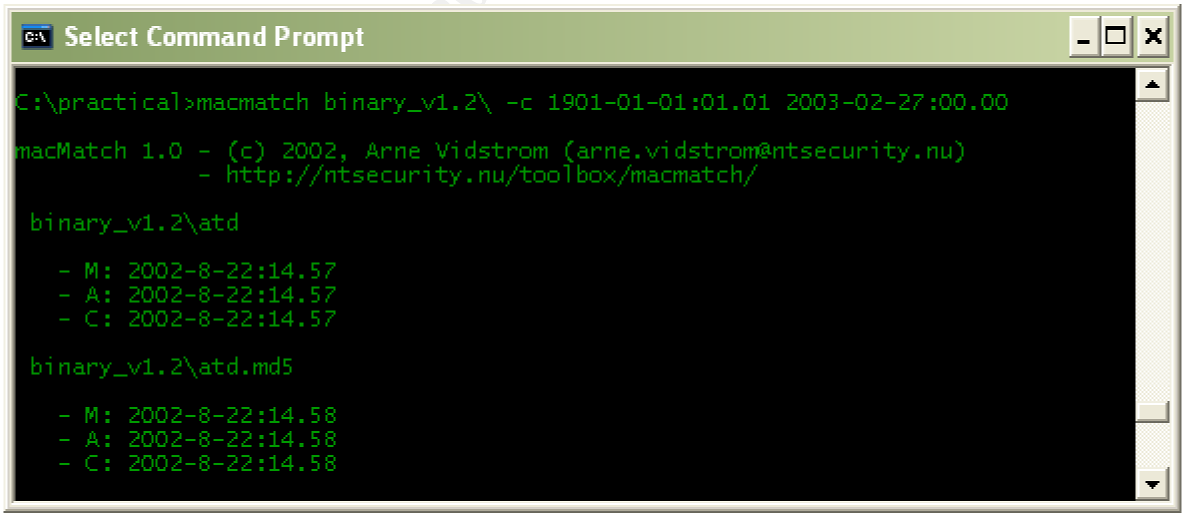
Next the newly compiled mac-robber binary is executed in order to see if the action of execution alters any of the MAC timestamps. To minimize the program’s activity, it is simply run with the –V parameter, which displays its version and author information rather than performing its primary purpose.

As the last step in this test, the stat tool is again used to display the current MAC times of the mac-robber binary after it has been executed.

The modify and change timestamps are unchanged, only the access timestamp has been updated and set to the time the program was executed.

```
# ./mac-robber –V

mac-robber information:

   version: 1.0

   author: Brian Carrier, @stake Inc.

   url: http://www.atstake.com/research/tools

[root@localhost mac-robber-1.00]# stat mac-robber
Access: Sun Apr  6 22:53:36 2003
Modify: Sun Apr  6 22:52:41 2003
Change: Sun Apr  6 22:52:41 2003
```

This demonstrates that given the MAC times for the binary file, as originally received, were all 2002-8-22:14:57 it is reasonable to conclude that one explanation for the identical timestamps is that this copy had never been executed. This, of course, makes the assumption that none of the MAC times were inappropriately altered.



```
C:\practical>macmatch binary_v1.2\ -c 1901-01-01:01.01 2003-02-27:00.00

macMatch 1.0 - (c) 2002, Arne Vidstrom (arne.vidstrom@ntsecurity.nu)
             - http://ntsecurity.nu/toolbox/macmatch/

 binary_v1.2\atd

   - M: 2002-8-22:14.57
   - A: 2002-8-22:14.57
   - C: 2002-8-22:14.57

 binary_v1.2\atd.md5

   - M: 2002-8-22:14.58
   - A: 2002-8-22:14.58
   - C: 2002-8-22:14.58
```

The analysis host and the monitoring host were to a 10 MB hub, with no connections (logical or physical) beyond these three devices so that the atd binary could be put into a network-enabled environment, allowing it to function as it was intended. The victim host (referred to originally as the disk-imaging host) had a fresh install of all workstation packages for Linux Redhat 7.2 (no updates) and the atd binary.

The monitor host ran tcpdump [8] to sniff the network during this portion of the analysis. Since there was no DHCP service provided, the IP information was manually configured.

The following commands were issued at the victim host. The parameters given to tcpdump would cause it to record all network activity for easy and detailed review after the test was completed. While tcpdump is usually used to capture TCP, UDP, and ARP traffic, meaning traffic to and from TCP port addresses, Loki2 takes advantage of ICMP, which does not use ports. Passing these parameters to tcpdump insured it would not exclude any type of packet from its logging.

```
# ifconfig eth0 172.24.0.10 netmask 255.255.255.0 up

# tcpdump –I eth0 -x -X -v -e -n -N -w tcpdump.atd
```

The following command was issued at the analysis host. This simply set its TCP/IP addressing for use on the lab LAN.

```
# ifconfig eth0 172.24.0.20 netmask 255.255.255.0 up
```

The ping [41] listing below confirmed the connectivity between the two hosts. Here the victim host pinged the analysis host. The analysis host is at 172.24.0.10; the host that will run atd (the victim host) is at 172.24.0.20.

```
11:39:18.505787 172.24.0.10 > 172.24.0.20: icmp: echo request (DF)
11:39:18.505875 172.24.0.20 > 172.24.0.10: icmp: echo reply
11:39:19.497970 172.24.0.10 > 172.24.0.20: icmp: echo request (DF)
11:39:19.497979 172.24.0.20 > 172.24.0.10: icmp: echo reply
11:39:20.497970 172.24.0.10 > 172.24.0.20: icmp: echo request (DF)
11:39:20.497978 172.24.0.20 > 172.24.0.10: icmp: echo reply
```

The analysis host then begins a port scan of the victim host.

```
11:50:27.141049 172.24.0.20.32769 > 172.24.0.10.25290: S 411674623:411674623(0) win 5840
<mss 1460,sackOK,timestamp 641111 0,nop,wscale 0> (DF)
11:50:27.141235 172.24.0.20.32770 > 172.24.0.10.57887: S 407713244:407713244(0) win 5840
<mss 1460,sackOK,timestamp 641111 0,nop,wscale 0> (DF)
11:50:27.141299 172.24.0.20.32771 > 172.24.0.10.44946: S 419387513:419387513(0) win 5840
<mss 1460,sackOK,timestamp 641111 0,nop,wscale 0> (DF)
11:50:27.142520 172.24.0.10.25290 > 172.24.0.20.32769: R 0:0(0) ack 411674624 win 0 (DF)
11:50:27.142585 172.24.0.10.57887 > 172.24.0.20.32770: R 0:0(0) ack 407713245 win 0 (DF)
11:50:27.142652 172.24.0.10.44946 > 172.24.0.20.32771: R 0:0(0) ack 419387514 win 0 (DF)
```

These tcpdump [8] listings confirmed the connectivity between the two hosts. In fact, the same ping and port scans were launched from the victim host to verify that the processes were in fact bi-directional.

The actions the program takes

To begin seeing how this program would actually behave, the program would be put into execution in an environment the program would consider “normal”. This would be accomplished with the strace [22] tool.

The strace [22] program was run on the host that would next launch atd, and tcpdump was run on the monitor host. The strace program acts as a wrapper around another program in order to logs the activities of the wrapped program.

As the strace home page notes;

> *strace is a system call trace, i.e. a debugging tool which prints out a trace of all the system calls made by a another process/program. The program to be traced need not be recompiled for this, so you can use it on binaries for which you don't have source.*
>
> *System calls and signals are events that happen at the user/kernel interface. A close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions* (strace 22)

Copies of the /etc and /usr/sbin directories was made, expecting that any malware activity, if atd is malware, would likely "touch" or alter these files. The grave-robber program was also run to create a timeline of each file on the analysis system. The grave-robber program will be used later to determine which files were accessed while atd was executing. It is not expected that this copy will be used, but it might be helpful to have this information available as the analysis progresses and these timestamps are altered.

Before launching atd, its mode had to be set to allow execution. The mode was changed from 0666 (rw-rw-rw-) to 700 (rwx------) allowing the file owner (root) to read, write and execute the file.

```
# chmod -e 700 atd
```

Finally, strace was run with the atd host. After all the information believed necessary had been collected, the kill command was used from another console (tty2) to terminate the atd process.

The syntax for atd was suggested in one of the strings findings (lokid -p (i|u) [-v (0|1)]), and then confirmed by the information on the hackerproof.org site (hackerproof.org/technotes/backdoors/index.html).

```
             Usage:

             1. Server-Side (= Target System)
               ~#./lokid -p i -v 1

             2. Client-Side (= Attacker's System)

     example: ~#./loki -d target_ip -p I -v 1 -t 3
```

```
# strace -c -ff -o atd.strace -r -s 1024 -v -x ./atd –p i –v 1
LOKI2  route [© 1997 guild corporation worldwide]
Process 1083 attached
Process 1083 detached
```

Looking into the /proc/1083 directory, I was able to find the details of strace, now that it was in execution and resident in memory. While strace was running as process 1083, atd was independently executing as process 1085.

```
# cat /proc/1083/maps
08048000-08063000 r-xp 00000000 03:03 344963     /usr/bin/strace
08063000-0806a000 rw-p 0001a000 03:03 344963     /usr/bin/strace
0806a000-0807e000 rwxp 00000000 00:00 0
40000000-40012000 r-xp 00000000 03:03 2109413    /lib/ld-2.2.93.so
40012000-40013000 rw-p 00012000 03:03 2109413    /lib/ld-2.2.93.so
40013000-40014000 rw-p 00000000 00:00 0
40021000-40022000 rw-p 00000000 00:00 0
42000000-42126000 r-xp 00000000 03:03 1259110    /lib/i686/libc-2.2.93.so
42126000-4212b000 rw-p 00126000 03:03 1259110    /lib/i686/libc-2.2.93.so
4212b000-4212f000 rw-p 00000000 00:00 0
bfffb000-c0000000 rwxp ffffc000 00:00 0
```

The strace tool provided a detailed activity log of atd while in execution, and that information will be analyzed in the next section.

As it turned out, tcpdump did not record any traffic related to the activity of atd while execution. This would make sense if atd is Loki2d, a daemon (loki**d**) intended to listen for connection requests rather than reaching out and indicating sessions. Even if there had been no expectation that atd would actively access the network, having a sniffer running while atd is in execution would be wise just in case the program, network oriented or not, was programmed to "phone home."

Forensic Details

Examining the services on this host, having just executing atd, the atd program has been loaded as a daemon and is ready to provide service when properly requested.

The ps tool clearly identifies the ./atd program running adjacent to strace. The pid 1085 can be used from this point to identify the program in process.

```
# ps -aux | grep atd
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root      1083  0.0  0.1  1532  528 tty1     S    13:01   0:00 strace -c -ff -o
root      1085  0.0  0.0   872  312 ?        S    13:01   0:00 ./atd -p i -v 1
```

When the network sockets are reported (netstat -anp), the atd program is report as process 1085, with a raw IP socket open at port 255. There is no "state" with raw mode or UDP, so the state field is often left blank. In this case, the "lack of state" is labeled "7."

```
# netstat –anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:515             0.0.0.0:*               LISTEN      645/lpd Waiting
tcp        0      0 0.0.0.0:7100            0.0.0.0:*               LISTEN      677/xfs
raw        0      0 0.0.0.0:1               0.0.0.0:*               7           1085/atd
raw        0      0 0.0.0.0:255             0.0.0.0:*               7           1085/atd
```

What other files are used when the program is executed or implemented?

While atd is in execution, the lsof (list open files) reports the files that strace and atd have open. In the case of atd, a couple of code libraries, a couple of terminal devices (tty1), and a couple of raw devices (the network ports) are open. If lsof had run at the moment atd was accessing passwd, group, localtime, or any other file it is coded to access, that file(s) would also be listed here. This illustrates the need to look at a condition from different points of view and "triangulate" a more complete and accurate picture of the condition.

```
# lsof
(all but strace and atd omitted)
Proto RefCnt Flags       Type       State        I-Node PID/Program name     Path

strace    1083 root  cwd    DIR       3,3    4096  1635831 /root/practical/atd/bin
strace    1083 root  rtd    DIR       3,3    4096        2 /
strace    1083 root  txt    REG       3,3  152927   344963 /usr/bin/strace
strace    1083 root  mem    REG       3,3   87341  2109413 /lib/ld-2.2.93.so
strace    1083 root  mem    REG       3,3 1395734  1259110 /lib/i686/libc-2.2.93.so
strace    1083 root   0u    CHR       4,1            71796 /dev/tty1
strace    1083 root   1u    CHR       4,1            71796 /dev/tty1
strace    1083 root   2u    CHR       4,1            71796 /dev/tty1
strace    1083 root   3w    REG       3,3       0  1636081 /root/practical/atd/bin/atd.strace
strace    1083 root   4w    REG       3,3       0  1636084 /root/practical/atd/bin/atd.strace.1085
atd       1085 root  cwd    DIR       3,3    4096   212577 /tmp
atd       1085 root  rtd    DIR       3,3    4096        2 /
atd       1085 root  txt    REG       3,3   15348   264621 /root/practical/atd/bin/atd
atd       1085 root  mem    REG       3,3   25386  2109705 /lib/ld-linux.so.1.9.5
atd       1085 root  mem    DEL       0,4            98307 /SYSV0000052e
atd       1085 root  mem    REG       3,3  699832   278524 /usr/i486-linux-libc5/lib/libc.so.5.3.12

Below the tty1 device is referenced.
atd       1085 root   1u    CHR       4,1            71796 /dev/tty1
atd       1085 root   2u    CHR       4,1            71796 /dev/tty1

Below, we see the information netstat reported earlier. the "raw" sockets used by atd and
launched by "root" are assigned ports 1 (:0001) and 255 (:00ff), both in state "07".

atd       1085 root   3u    raw                      14497 00000000:0001->00000000:0000 st=07
atd       1085 root   4u    raw                      14498 00000000:00FF->00000000:0000 st=07
```

How is the file system affected by the execution of the program?

When atd/Lokid is executed, it makes a few changes to local files. Some of the changes are side effects, system logs are updated, and access timestamps are updated on the files this program uses.

The major alterations made to the file system include:

- The program's actual presence as a file, and
- The process and other local system activity that is logged.

Does the program use, manipulate, or reference any other system files?

Take a detailed look at any MAC time changes that occurred since atd was executed.

The strace program does a great job of recording information about a program's activity, including file access, while in execution.

The atd program is put into execution with the parameters –p I –v 1.

```
0.000000 execve("/usr/sbin/atd", ["atd", "-p", "i", "-v", "1"], [/* 27 vars */]) = 0
0.002151 uname({sysname="Linux", nodename="localhost.localdomain", release="2.4.7-10",
version="#1 Thu Sep 6 17:21:28 EDT 2001", machine="i586"}) = 0
```

brk sets the end of the data segment to the value specified by *end_data_segment*, when that value is reasonable, the system does have enough memory and the process does not exceed its max data size. 0.000775 brk(0)   = 0x804c58c

Two files, ld.so.preload and ld.so.cache are opened read-only, though preload is not found.

```
0.000326 open("/etc/ld.so.preload", O_RDONLY) = -1 ENOENT (No such file or directory)
0.000394 open("/etc/ld.so.cache", O_RDONLY) = 3
```

According the fsstat manpage, "… fstat, that utilize a 64-bit integer status array to return information about long files." The structure of the returned information is:

```
    dev_t       st_dev;      /* device */
    ino_t       st_ino;      /* inode */
    mode_t      st_mode;     /* protection */
    nlink_t     st_nlink;    /* number of hard links */
    uid_t       st_uid;      /* user ID of owner */
    gid_t       st_gid;      /* group ID of owner */
    dev_t       st_rdev;     /* device type (if inode device) */
    off_t       st_size;     /* total size, in bytes */
    unsigned long st_blksize;  /* blocksize for filesystem I/O */
    unsigned long st_blocks;   /* number of blocks allocated */
    time_t      st_atime;    /* time of last access */
    time_t      st_mtime;    /* time of last modification */
    time_t      st_ctime;    /* time of last change */
0.000261 fstat64(3, {st_dev=makedev(3, 67), st_ino=227379, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=176, st_size=82021,
st_atime=2003/03/03-15:02:55, st_mtime=2003/02/28-06:18:41, st_ctime=2003/02/28-06:18:41})
= 0
```

According to the mmap manpage, "The mmap function asks to map *length* bytes starting at offset *offset* from the file (or other object) specified by the file descriptor *fd* into memory, preferably at address *start*. This latter address is a hint only, and is usually specified as 0. The actual place where the object is mapped is returned by mmap. The *prot* argument describes the desired memory protection (and must not conflict with the open mode of the file)."

```
0.001301 old_mmap(NULL, 82021, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40016000
```

The ld.so.cache file is now closed,

```
0.000303 close(3)                = 0
```

and lib.so.6 is open read-only.

```
0.000262 open("/lib/libc.so.6", O_RDONLY) = 3
```

1 kb of material is read from lib/libc.so.6 and represented below in hex notation.

```
0.000280 read(3, "\x7f\x45\x4c
```

A large amount of hex material has been omitted for readability.

```
        \x00\x53\x05"..., 1024) = 1024

0.006136 fstat64(3, {st_dev=makedev(3, 67), st_ino=305874, st_mode=S_IFREG|0755,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=11192, st_size=5716491,
st_atime=2003/03/03-15:02:55, st_mtime=2001/09/04-15:36:31, st_ctime=2003/02/28-05:42:53})
= 0

0.000536 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4002b000
0.000382 old_mmap(NULL, 1264328, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x4002c000
```

According to the mprotect manpage, “mprotect controls how a section of memory may be accessed. If an access is disallowed by the protection given it, the program receives a SIGSEGV.”

```
0.000295 mprotect(0x40158000, 35528, PROT_NONE) = 0

0.000213 old_mmap(0x40158000, 20480, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x12b000) = 0x40158000
0.000464 old_mmap(0x4015d000, 15048, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4015d000
0.000315 close(3)                = 0
```

The memory mapping performed by mmap above is deallocated in the next line.

```
0.002492 munmap(0x40016000, 82021) = 0

0.000653 brk(0)                  = 0x804c58c
0.000214 brk(0x804c9a4)          = 0x804c9a4
0.000192 brk(0x804d000)          = 0x804d000
```

According to the socket manpage, “Socket creates an endpoint for communication and returns a descriptor. The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication.”

```
0.000262 socket(PF_UNIX, SOCK_STREAM, 0) = 3
```

Initiate a connection on a socket.

```
0.000332 connect(3, {sin_family=AF_UNIX, path="/var/run/.nscd_socket"}, 110) = -1 ENOENT
(No such file or directory)
0.000442 close(3)                = 0
```

Open and read the nsswitch.conf file.

```
0.000375 open("/etc/nsswitch.conf", O_RDONLY) = 3
0.000362 fstat64(3, {st_dev=makedev(3, 67), st_ino=225418, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=8, st_size=1750,
st_atime=2003/03/03-15:02:49, st_mtime=2003/02/28-06:19:25, st_ctime=2003/02/28-06:19:25})
= 0
0.000563 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
```

The nsswitch.conf file is opened and 1750 bytes are read.

```
0.000335 read(3, "#\n# /etc/nsswitch.conf\n
```

A large amount of hex material has been omitted for readability.

```
      "..., 4096) = 1750

0.002063 brk(0x804e000)          = 0x804e000
0.000350 read(3, "", 4096)       = 0
0.000305 close(3)                = 0

0.000208 munmap(0x40016000, 4096)  = 0
```

Again, open the ld.so.cache file for more material.

```
0.000419 open("/etc/ld.so.cache", O_RDONLY) = 3
0.000351 fstat64(3, {st_dev=makedev(3, 67), st_ino=227379, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=176, st_size=82021,
st_atime=2003/03/03-15:02:55, st_mtime=2003/02/28-06:18:41, st_ctime=2003/02/28-06:18:41})
= 0
0.000531 old_mmap(NULL, 82021, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40016000
0.000283 close(3)              = 0
```

Open and read libnss_files.so.2

```
0.000252 open("/lib/libnss_files.so.2", O_RDONLY) = 3
0.000279 read(3, "\x7f\x45\
        A large amount of hex material has been omitted for readability.
        x00\x21\x00"..., 1024) = 1024
0.005844 fstat64(3, {st_dev=makedev(3, 67), st_ino=305899, st_mode=S_IFREG|0755,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=512, st_size=255971,
st_atime=2003/03/03-15:02:55, st_mtime=2001/09/04-15:36:16, st_ctime=2003/02/28-05:42:54})
= 0
0.000594 old_mmap(NULL, 41408, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x40161000
0.000302 mprotect(0x4016a000, 4544, PROT_NONE) = 0
0.000217 old_mmap(0x4016a000, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x8000) = 0x4016a000
0.000413 old_mmap(0x4016b000, 448, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4016b000
0.000308 close(3)              = 0
0.000478 munmap(0x40016000, 82021) = 0
```

This is disturbing! The passwd file is opened. This is the file that contains most of the control information for every account on this system.

```
0.000397 open("/etc/passwd", O_RDONLY) = 3
0.000398 fcntl64(0x3, 0x1, 0, 0x1) = 0
0.000220 fcntl64(0x3, 0x2, 0x1, 0x1) = 0
0.000269 fstat64(3, {st_dev=makedev(3, 67), st_ino=227385, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=8, st_size=1272,
st_atime=2003/03/03-15:02:49, st_mtime=2003/02/28-06:19:25, st_ctime=2003/02/28-06:19:25})
= 0
0.000660 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
```

The passwd file is read. In this short example, the root and bin accounts are read. Not that the second field in each record is an “x”, indicating the password hashes are stored in another file, shadow.

```
0.000311 read(3, "root:x:0:0:root:/root:/bin/bash\nbin:x:1:1:bin:/bin:/sbin/nologin\n
        The remainder of /etc/passed has been omitted for readability.
        ", 4096) = 1272
0.001585 close(3)              = 0
0.000216 munmap(0x40016000, 4096)  = 0
```

Again, open a socket.

```
0.000315 socket(PF_UNIX, SOCK_STREAM, 0) = 3
0.000295 connect(3, {sin_family=AF_UNIX, path="/var/run/.nscd_socket"}, 110) = -1 ENOENT
(No such file or directory)
0.000378 close(3)              = 0
```

Next the group file is read. This will tell the program when user account are members of which groups.

```
0.000321 open("/etc/group", O_RDONLY) = 3
0.000287 fcntl64(0x3, 0x1, 0, 0x1) = 0
0.000210 fcntl64(0x3, 0x2, 0x1, 0x1) = 0
0.000216 fstat64(3, {st_dev=makedev(3, 67), st_ino=227389, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=8, st_size=561, st_atime=2003/03/03-
15:02:49, st_mtime=2003/02/28-06:19:25, st_ctime=2003/02/28-06:19:25}) = 0
0.000533 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
0.000301 read(3, "root:x:0:root\nbin:x:1:root,bin,daemon\n
        The remainder of /etc/group has been omitted for readability.
        ", 4096) = 561
0.000896 close(3)                = 0
0.000212 munmap(0x40016000, 4096)  = 0
```

According to the geteuid manpage, "geteuid returns the effective user ID of the current process. The effective ID corresponds to the set ID bit on the file being executed. Both the user and group ID# are queried."

```
0.000278 geteuid32()              = 0
0.000215 getegid32()              = 0
```

According to the setreuid manpage, "setreuid, setregid - set real and/or effective user or group ID setreuid sets real and effective user IDs of the current process. Unprivileged users may only set the real user ID to the real user ID or the effective user ID, and may only set the effective user ID to the real user ID, the effective user ID or the saved user ID."

```
0.000211 setregid32(0, 0x2)       = 0
0.000220 setreuid32(0, 0x2)       = 0
0.000401 brk(0x8051000)           = 0x8051000
```

According to the time manpage, "time returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds. If (*t*) is non-NULL, the return value is also stored in the memory pointed to by *t*."

```
0.000369 time([1046721775])       = 1046721775
```

And now the localtime file is read. The program is continuing to gather information about its current environment.

```
0.000290 open("/etc/localtime", O_RDONLY) = 3
0.000343 fstat64(3, {st_dev=makedev(3, 67), st_ino=225417, st_mode=S_IFREG|0644,
st_nlink=1, st_uid=0, st_gid=0, st_blksize=4096, st_blocks=8, st_size=1267,
st_atime=2003/03/03-15:02:55, st_mtime=2003/02/28-06:19:24, st_ctime=2003/02/28-06:19:24})
= 0
0.000538 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
```

The local time file is read, and represented here in hex notation.

```
0.000306 read(3, "\x54\x5a\x69\
        A large amount of hex material has been omitted for readability.
        \x70\x54\x4c"..., 4096) = 1267
0.005855 close(3)                = 0
0.000220 munmap(0x40016000, 4096)  = 0

0.000408 getpid()                = 1453

0.000337 rt_sigaction(SIGPIPE, {0x4010f1c8, [], 0x4000000}, {SIG_DFL}, 8) = 0
0.000392 socket(PF_UNIX, SOCK_DGRAM, 0) = 3
```

```
0.000287 fcntl64(0x3, 0x2, 0x1, 0x40153d87) = 0

A connection is made to the /dev/log file. Then an entry is written to the log file.
0.000224 connect(3, {sin_family=AF_UNIX, path="/dev/log"}, 16) = 0
0.000313 send(3, "<75>Mar  3 15:02:55 atd[1453]: unknown option", 45, 0) = 45
0.000881 rt_sigaction(SIGPIPE, {SIG_DFL}, NULL, 8) = 0
0.000464 _exit(1)                = ?
```

MAC time analysis

Now that the suspect program has been executed, a review of the MAC times report for the entire system will pinpoint the files that were accessed while atd was in execution.

After atd had been in execution for a few moments, it was terminated and grave-robber was again run on the entire system. Once grave-robber had collected the system information, mactime was used to extract a listing of the files throughout the system organized by access time.

```
The atd program is executed within the strace program.
Thu Apr 24 2003 11:14:09     1931 .a. -rw-r--r-- 0       0       1635984
/root/practical/atd/bin/atd.strace.1085
Thu Apr 24 2003 11:15:05     1931 m.c -rw-r--r-- 0       0       1635984
/root/practical/atd/bin/atd.strace.1085
Thu Apr 24 2003 11:18:11   152927 .a. -rwxr-xr-x 0       0       344963   /usr/bin/strace
                             1931 .a. -rw-r--r-- 0       0       1635985  /root/practical/atd/bin/atd.strace. 1085
                             4096 m.c drwxr-xr-x 0       0       1635831  /root/practical/atd/bin

The operating system validates root's permissions to execute this file.
Thu Apr 24 2003 11:18:26      643 .a. -rw-r--r-- 0       0       1389952  /etc/pam.d/system-auth
                            15294 .a. -rwxr-xr-x 0       0       752231   /lib/security/pam_limits.so
                             8836 .a. -rwxr-xr-x 0       0       752242   /lib/security/pam_securetty.so
                            14073 .a. -rwxr-xr-x 0       0       752225   /lib/security/pam_env.so
                              427 .a. -rw-r--r-- 0       0       1389990  /etc/pam.d/login
                           183149 .a. -rwxr-xr-x 0       0       359762   /usr/lib/libglib-1.2.so.0.0.10
                            12637 .a. -rwxr-xr-x 0       0       2109495  /lib/libpam_misc.so.0.75
                            52235 .a. -rwxr-xr-x 0       0       752222   /lib/security/pam_console.so
                            13655 .a. -rwxr-xr-x 0       0       752244   /lib/security/pam_stack.so
                              210 .a. -rw-r--r-- 0       0       1389951  /etc/pam.d/other
                            63761 .a. -rwxr-xr-x 0       0       359754   /usr/lib/libcrack.so.2.7
                             7046 .a. -rwxr-xr-x 0       0       752237   /lib/security/pam_nologin.so
                            37114 .a. -rwxr-xr-x 0       0       2109494  /lib/libpam.so.0.75
                            25942 .a. -rwxr-xr-x 0       0       948482   /bin/login
                              114 .a. -rw------- 0       0       228949   /etc/securetty
                             5342 .a. -rwxr-xr-x 0       0       752224   /lib/security/pam_deny.so
                            15443 .a. -rwxr-xr-x 0       0       752223   /lib/security/pam_cracklib.so
Thu Apr 24 2003 11:18:31      719 .a. -rwxr-xr-x 0       0       719519   /etc/profile.d/colorls.sh
Five seconds into execution, atd seems to have referenced the gzip program.
                            63488 .a. -rwxr-xr-x 0       0       948496   /bin/gzip
This script, gnome-ssh-askpass.sh,  exports the ssh password
                               70 .a. -rwxr-xr-x 0       0       719919   /etc/profile.d/gnome-ssh-askpass.sh
Quit a lot of the rest reflects the login process.
                             4096 .a. drwxr-xr-x 0       0       719491   /etc/profile.d
                              756 .a. -rw-r--r-- 0       0       228943   /etc/inputrc
                              842 .a. -rw-r--r-- 0       0       228947   /etc/profile
                             8580 .a. -rwxr-xr-x 0       0       343474   /usr/bin/tput
```

```
               23537 .a. -rwxr-xr-x 0       0       343643   /usr/bin/setterm
Opening up a terminal enokes the motd banner.
                   0 .a. -rw-r--r-- 0       0       228944   /etc/motd
                 170 .a. -rwxr-xr-x 0       0       719520   /etc/profile.d/which-2.sh
Also five seconds into execution, atd references the egrep binary.
                  33 .a. -rwxr-xr-x 0       0       948437   /bin/egrep
utmp was updated. A later review using the who tool did not show anything of interest.
                4224 m.c -rw-rw-r-- 0       22      1095624  /var/run/utmp
                 223 .a. -rw-r--r-- 0       0       294986   /root/.bashrc
                2434 .a. -rw-r--r-- 0       0       228969   /etc/DIR_COLORS
                1418 .a. -rw-r--r-- 0       0       1439008  /etc/security/limits.conf
               36728 .a. -rwxr-xr-x 0       0       948472   /bin/stty
                4420 .a. -rwxr-xr-x 0       0       49123    /sbin/consoletype
                 190 .a. -rwxr-xr-x 0       0       719495   /etc/profile.d/glib2.sh
                 153 .a. -rwxr-xr-x 0       0       719647   /etc/profile.d/vim.sh
                 267 .a. -rwxr-xr-x 0       0       719563   /etc/profile.d/less.sh
               29266 .a. -rwxr-xr-x 0       0       343532   /usr/bin/wc
               30424 .a. -rwxr-xr-x 0       0       948493   /bin/setfont
                1497 .a. -rw-r--r-- 0       0       228934   /etc/bashrc
              823296 m.c -rw-rw-r-- 0       22      670841   /var/log/wtmp
            19136220 m.c -r-------- 0       0       670436   /var/log/lastlog
                  80 .a. -rw-r--r-- 0       0       245998   /etc/sysconfig/i18n
                1426 .a. -rwxr-xr-x 0       0       948494   /bin/unicode_start
                4900 .a. -rwxr-xr-x 0       0       948491   /bin/kbd_mode
               50616 .a. -rwxr-xr-x 0       0       948490   /bin/dumpkeys
               75416 .a. -rwxr-xr-x 0       0       948492   /bin/loadkeys
                2862 .a. -rw-r--r-- 0       0       1439009  /etc/security/pam_env.conf
                 279 .a. -rw-r--r-- 0       0       295016   /root/.bash_profile
Thu Apr 24 2003 11:18:37     4096 .a. drwxr-x--- 0       0       294337   /root
Here the lsof and ps reports were created.
Thu Apr 24 2003 11:18:58    26206 m.c -rw-r--r-- 0       0       264634
/root/practical/atd/strace.atd.lsof
Thu Apr 24 2003 11:19:17     2505 m.c -rw-r--r-- 0       0       264635
/root/practical/atd/stace.atd.ps
Thu Apr 24 2003 11:19:30     1931 m.c -rw-r--r-- 0       0       1635985
/root/practical/atd/bin/atd.strace.1085
                               84 m.c -rw-r--r—0       0       1636081
/root/practical/atd/bin/atd.strace
```

Does this file include any additional “leads” for further investigation?

Other than the strings that were found in the binary, there were no other items that seemed to provide any clue for further investigation. These were, however, enough to track down the source code along with two articles [20] explaining the program in detail.

Program Identification

By searching Google.com for some of the strings found in the atd binary, several promising references immediately came up.

The strings used in the search were:

1. lokid: cannot locate client entry in database;
2. lokid: unsupported or unknown command string;

3. lokid: client <%d> requested an all kill; and
4. route [(c) 1997 guild corporation worldwide]

While the same was true of many other hits, the URL below caught my attention, since it referred to a “tunneling program.”

- http://secinf.net/unix_security/LOKI2__informationtunneling_program_and_description.html

That URL quickly pointed me to the Phrack.org pages that had originally introduced the loki suite of covert channel tools. Other sites helped add to the description of loki and its uses and operation.

Recreating this binary

I was unable to recreate an exact duplicate of the atd, a.k.a. Loki2d. While atd was successfully compiled, neither the file size or md5 hash matched the original.

So, I turned to other techniques in an attempt to quantify the likelihood that the binary was in fact a copy of Loki2d.

First, the strings found in the atd binary were compared with the source code from Phrack Magazine Volume 7, Issue 51 September 01, 1997, article 06 of 17, "LOKI2 (the implementation)."[20]

To do this, a list of sorted lines from lokid.c was created and then edited to remove the “code” strings leaving only the more readable stings. That left the following 28 strings:

```
 [ lokid.c ]
 [fatal] could not signal parent
 [fatal] could not signal process group
 [fatal] forking error
 [non fatal] truncated write
1996/7 Guild Corporation Worldwide      [daemon9]
Cannot set IP_HDRINCL socket option
LOKI2
lokid: %s
lokid: Cannot add key
lokid: client <%d> added to list [%d]
lokid: client <%d> freed from list [%d]
lokid: client <%d> requested a protocol swap
lokid: client <%d> requested an all kill
lokid: computing shared secret
lokid: done.
lokid: extracting 128-bit blowfish key
lokid: packet read %d bytes, type:
lokid: popen
lokid: public key submission and request : %s <%d>
lokid: submiting my public key to client
lokid: transport protocol changed to %s
mfd: %d
Raw IP socket:
sending L_QUIT: <%d> %s
sending protocol update: <%d> %s [%d]
Unknown transport
```

```
v:p:
```

A quick walk-through confirmed that each of these 28 strings existed in the atd binary. In fact, the source code showed that each string harvested from atd was used in the fprintf()[20] function in the source code.

This gave me a high degree of confidence that atd was, in fact, a compiled copy of loki2d.c

According to Richard Ginski's GCFA practical [52] the version of loki available from Phrack will not compile properly, but there is a version at hackerproof.org that will.

Next, the atd source codes from phrack and from hackerproof.org were compared to help determine how identical these two files were.

Below, the difference between the lokid.c code files from Phrack.org [20] and hackerproof.org [61] are illustrated.

```
# diff --side-by-side --suppress-common-lines –W 60 lokid.hackproof/ lokid-Phrack/
```

```
Only in loki-hackerproof/: client_db.o
Only in loki-hackerproof/: crypt.o
Only in loki-hackerproof/: loki

diff --side-by-side –suppress-common-lines -W 60 loki-hackerproof/loki.c loki-Phrack/loki.c
fprintf(stderr, "\n[ |          fprintf(stderr, "\n[
Only in loki-hackerproof/: lokid

diff --side-by-side –suppress-common-lines -W 60 loki-hackerproof/lokid.c loki-Phrack/lokid.c
fprintf(stderr, "\n[ |          fprintf(stderr, "\n[
err_ |        err_exit
Only in loki-hackerproof/: lokid.o

diff --side-by-side –suppress-common-lines -W 60 loki-hackerproof/loki.h loki-Phrack/loki.h
#include <linux/ip.h>           <
                                >     #include <linux/ip.h>
                                >     #include <linux/signal.h>
#define FIX_LEN(n)  (n)         |     #define FIX_LEN(n)  (x)
#ifdef NET3                     |     #ifdef NET3
if      (ldg.payload[0]  == |         if      (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else if (ldg.payload[0]   == |        else if (ldg.payload[0]   ==
else |    else
                                >     if      (prot == IPPROTO_ICM
                                >     else
if (prot == IPPROTO_ICMP) fp <
                                <
Only in loki-hackerproof/: loki.o
```

```
diff --side-by-side –suppress-common-lines -W 60 loki-hackerproof/Makefile loki-
Phrack/Makefile
#CRYPTO_TYPE                 =  |    CRYPTO_TYPE                =
CRYPTO_TYPE                  =  |    #CRYPTO_TYPE               =
#   Addedum                  |      #   It is safe to leave this
NET3    =   |     NET3            =
DEBUG   =   |   DEBUG   =
        @( cd .. ; tar cvf l |                 @( cd .. ; tar cvf l

Common subdirectories: loki-hackerproof/md5 and loki-Phrack/md5
Only in loki-hackerproof/: pty.o
Only in loki-hackerproof/: shm.o

diff --side-by-side –suppress-common-lines -W 60 loki-hackerproof/surplus.c loki-
Phrack/surplus.c
 * LOKI                      |       * LOKI2
                             <
Only in loki-hackerproof/: surplus.o
```

From this report, several conclusions can be drawn.

- According to the strings comparison, there are only minor cosmetic differences.
- According to lsof, loki opened the same files as atd.
- According to netstat, loki opened the same type of raw socket on the same port as atd.
- According to ps, loki ran with the same settings as atd.

Based on these findings, I am confident in asserting that this unknown binary named atd is an instance of the lokid program.

Conclusion

This program, atd, is a copy of the loki2d daemon that provides the backend of a covert ICMP communications channel.

This program allows a person to direct the operation of a remote (likely compromised) system while undetected by most network based security devices unless they are configured to pay special attention to what is otherwise considered normal ICMP traffic.

The person who placed this file on a company system intended to make use of that system while likely is circumventing any network security components. Knowing the content and logical position in the network would likely reveled the intended use of this covert channel.

Legal Implications

I was unable to prove that this binary had been executed. Neither the MAC times, nor the metadata within the zip archives, support any indication the file had been accessed either for execution or any other purpose since it was created. It is more likely that either the actual MAC times were altered while moving the file between file systems, or they were intentionally altered to destroy their value as evidence.

At the very least, this is a violation of enterprise policy. That policy states that only software approved by the technical support group can be installed on a company system, and then only by members of the technical support group or by vendors under the supervision of a member of the technology support group.

While a good case might be made for a specific use of the functions provided by a tool like loki within a business environment, approval would be denied on the basis that as a covert listener released to the general public, loki would be a desirable target for an intruder, and one that could not adequately be controlled to insure only authorized access. Other similar tools that enforce client authentication are available. The question of vulnerabilities within the program itself would be another reason use of loki would be denied. Vulnerabilities and bugs are not likely to be "patched" by the developer in a timely fashion.

The article in Phrack which announced loki described its purpose as;

> "It [loki] can be used as a backdoor into a system by providing a covert method of getting commands executed on a target machine. It can be used as a way of clandestinely leeching information off of a machine. It can be used as a covert method of user-machine or user-user communication." (Phrack 20)

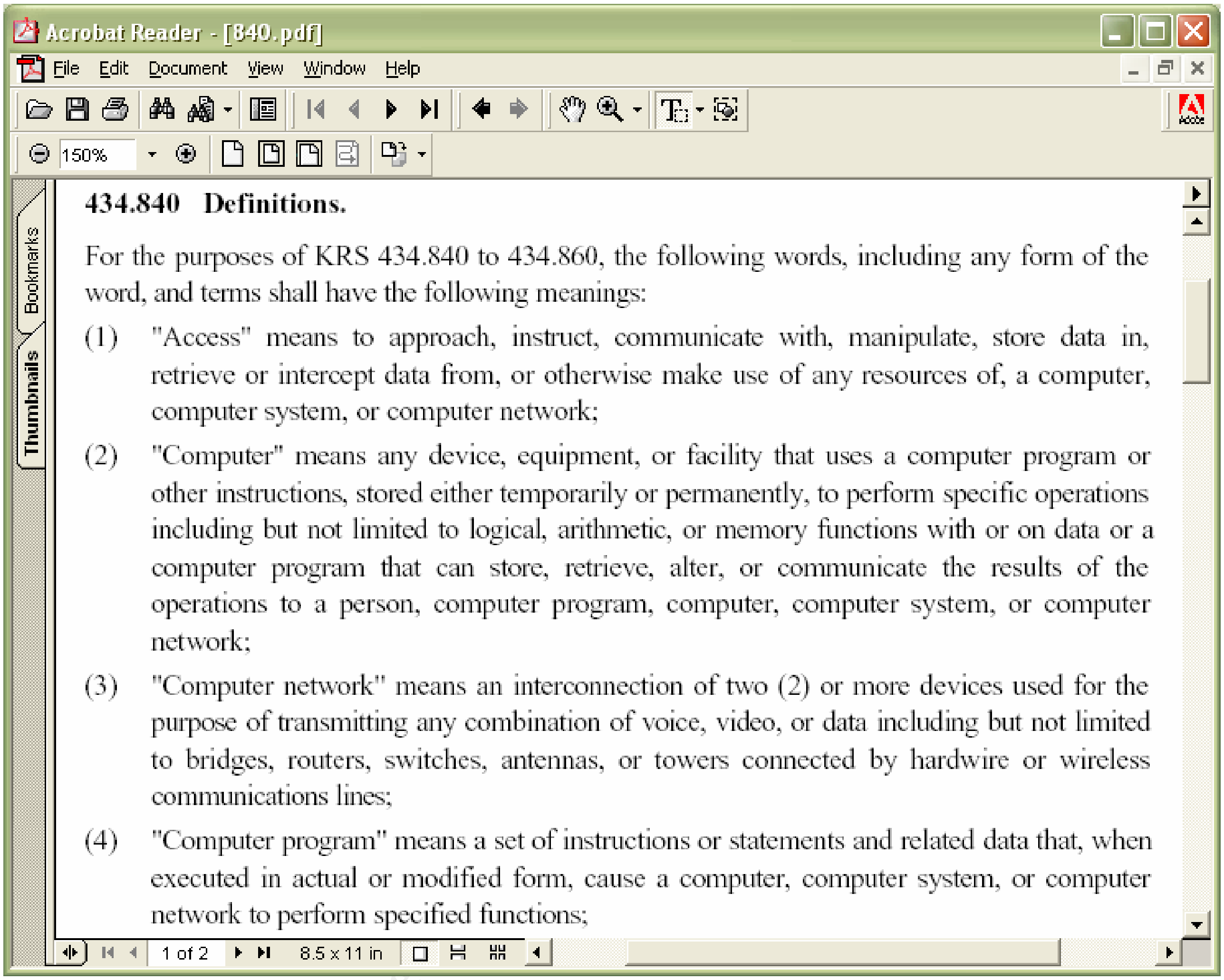
Loki could be used either to;

1. Submit covert commands to be executed by a remote host, or
2. to allow two people to covertly communicate in real-time.

In this case, the only reason a person would place this program on a system is if they intended to covertly communicate through a network, which implies unauthorized access. Since a covert communications channel is "hidden" from the communications infrastructure rather than the hosts participating in the communications, the effect of lokid is on the network rather than the network hosts. KRS 434.840 and KRS 434.845 (KRS 53) together make it clear that the issue of "access" is not limited to the end-point host, but includes the network infrastructure.

The argument could be made that the lock-pick principle applies here. "The only reason a person would have possession of a lock picking tool is that they intended to open a lock without authorization."

Even if the person using loki were authorized to use both hosts, using lokid to would seem to violate KRS 434.845 (KRS 53) in that loki is used to "obtain services [of a computer network] by means of fraudulent representations." The violation does not have to do with the hosts themselves, but with the use through "fraudulent representations" of the network(s) that connect them.


Acrobat Reader - [840.pdf]
File Edit Document View Window Help
150%
Bookmarks
Thumbnails
1 of 2
8.5 x 11 in

434.840 Definitions.

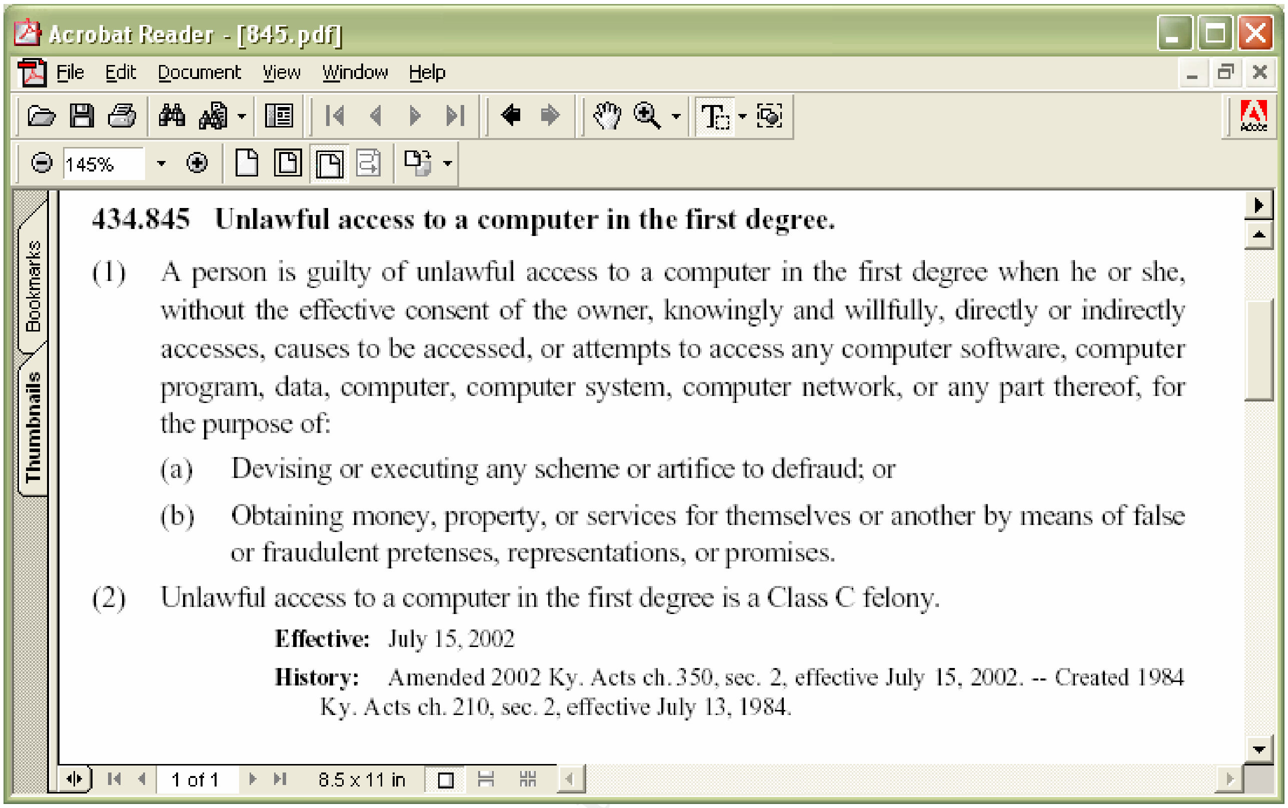
For the purposes of KRS 434.840 to 434.860, the following words, including any form of the word, and terms shall have the following meanings:

(1) "Access" means to approach, instruct, communicate with, manipulate, store data in, retrieve or intercept data from, or otherwise make use of any resources of, a computer, computer system, or computer network;

(2) "Computer" means any device, equipment, or facility that uses a computer program or other instructions, stored either temporarily or permanently, to perform specific operations including but not limited to logical, arithmetic, or memory functions with or on data or a computer program that can store, retrieve, alter, or communicate the results of the operations to a person, computer program, computer, computer system, or computer network;

(3) "Computer network" means an interconnection of two (2) or more devices used for the purpose of transmitting any combination of voice, video, or data including but not limited to bridges, routers, switches, antennas, or towers connected by hardwire or wireless communications lines;

(4) "Computer program" means a set of instructions or statements and related data that, when executed in actual or modified form, cause a computer, computer system, or computer network to perform specified functions;


Acrobat Reader - [845.pdf]
File Edit Document View Window Help
145%
Bookmarks
Thumbnails
1 of 1
8.5 x 11 in

434.845 Unlawful access to a computer in the first degree.

(1) A person is guilty of unlawful access to a computer in the first degree when he or she, without the effective consent of the owner, knowingly and willfully, directly or indirectly accesses, causes to be accessed, or attempts to access any computer software, computer program, data, computer, computer system, computer network, or any part thereof, for the purpose of:

(a) Devising or executing any scheme or artifice to defraud; or

(b) Obtaining money, property, or services for themselves or another by means of false or fraudulent pretenses, representations, or promises.

(2) Unlawful access to a computer in the first degree is a Class C felony.

Effective: July 15, 2002

History: Amended 2002 Ky. Acts ch. 350, sec. 2, effective July 15, 2002. -- Created 1984 Ky. Acts ch. 210, sec. 2, effective July 13, 1984.

Interview Questions

There are a few vectors the questioner can take, depending on the person being interviewed.

One vector is wrapping the questions in a Boy-I-Admire-Someone-Who-Can-Do-These-Things delivery. This allows a person to explain their perspective and capabilities regarding the topic. In this case, the topic is "covert communications daemons." While this is an appeal to the ego, the focus is more on camaraderie.

1. I don't know about you, but I am not comfortable with the way some of our network communications are sent across the wires in clear text. For that matter, it is too easy for some of the "computer nazis" around here to identify *private communications*. How would you suggest I get around that?
2. There are not many folks around here who are sharp enough to get a daemon running on a Linux host, let alone without crashing the system! I hear you have had some success doing that on your own. Can you give me any tips?

Another vector would be a direct appeal to the person's ego.

1. We found this atd program and need some help responding to the situation. Can you help?
2. My boss said you probably could not help, because you are not smart enough, but something tells me you can. What about it? If you help me; it proves you are ore capable than the boss things.

One other vector is the FUD (Fear, Uncertainty, and Doubt) factor.

1. This is serious. The boss is talking about calling the cops. If there is anything you know about this, I would rather we take care of it privately (and maybe recruit a new hacker to our security department).
2. This happened once before. We think you might know something about it and expect to know for certain from the evidence shortly. I can give you this one chance to help me. What can you tell me?
3. The boss really believes you are involved with this. I cannot promise anything; if you helped out by telling me what this is all about, and how we can "fix" it, he might not want to make a big deal of it. That is a promise I cannot make; but it is what I think.

Additional Information

Additional information on the analysis of unknown binary files is available at the following sites:

1. Phrack at www.Phrack.org;
2. The Honeynet project at www.honeynet.org especially reviewing the results of each "scan of the month", found at www.honeynet.org/scans/; and
3. "Analysis of 'the-binary'" available at laad.altervista.org/tuts/anbin.txt

Part 2, Option 1 - Perform forensic analysis on a system

Synopsis of Case Facts

This section describes the forensic analysis performed on two hard disks taken from a single desktop system.

A large number of PC class computers had been sent to surplus. Enterprise policy is clear and states each system must be effectively wiped of all electronic material, without respect to the type of material, before leaving the control of the department or sent to surplus.

It was suspected the wiping of the hard disks had not been properly performed. To verify the policy was both being followed and was effective, a forensic audit of a sample machine would be preformed. The goals were to answer the following questions:

1. Is confidential or proprietary information being unintentionally leaked through the surplus of system?
2. Do these systems show signs of having been compromised, either by an unauthorized intruder, or undetected malware (malicious software, viruses, or adware)?
3. And, do these systems show evidence of inappropriate use.

Time and resources permitted the analysis of only a single desktop system.

When the sample system was acquired, nothing was known of its history, contents, uses or operational condition. The system consisted of the desktop case and internal components. There was no keyboard, display, power cord, mouse, or other peripherals.

When selecting the sample host, each case was inspected. Specifically, the screws holding the case closed were inspected for signs of excessive ware (striped screws, missing screws, mismatched screws, screw driver marks on the case). The assumption was that this would indicate the case had been opened excessively and that the internal components were more likely to have been altered or physically accessed due to hardware failure, upgrading, or scavenging of parts. The boxes that showed this type of wear were excluded from selection.

Each potential sample system was opened to verify that it contained a hard disk(s) that “appeared” to be functional. Again, the internal screws were inspected for wear and the arrangement of wires and bus cables was checked for indications of “tinkering”. The age of the system, based on dates stamped on the case and components, was also taken into consideration. The assumption was the older the disk, the more use, and the larger the variety of uses, it was likely to have seen.

There seems to be a pattern where a host is initially used for more “critical or important” tasks; then as the host ages, it is moved to areas or tasks of decreasing importance. With that hypothesis, while a host may have been of minor “security interest” when taken out of service, it may unknowingly still contain highly sensitive material or artifacts (portions of files) from prior usage.

There was no requirement for other components, such as network adapters or removable media, since the intention was to move the hard disk(s) from this sample system to a trusted host that would be used to make a byte-for-byte identical copy of the entire disk. Any other components were of no value to the investigation.

Using these criteria, a desktop system was selected for forensic analysis.

Describe the system(s) you will be analyzing

There was no external indication of the type of operating system, file system, primary use or other descriptive information. An attached property tag did identify the owning department.

The system was acquired from a surplus property warehouse where it was found in a stack of about 100 PC cases of varying age, maker, model, and type (desktop, tower).

There were other similar stacks of hardware; all were similarly a mixture of desktop computers. As individual systems had been sent to surplus over time, they were gradually bundled into these stacks for bulk auction to the public or other disposal.

Most of these systems contained a single and relatively small hard disk, ranging in size from 800 MB to 4 GB. The system that was selected contained two physical hard disks, one configured as the IDE master and the other as the IDE slave. This offered the safeguard that, if one disk was not operational, the other might be.

Hardware

This system was labeled **Unit1**.

Suspect system hardware inventory
Analysis Case Identification – Unit1

```
Gateway P5-133 (133 MHz Pentium)
128 MB RAM
Desktop class machine
Inventory tag #081318
Internal 6x CDROM player
Internal 3.5" 1.44 MB high density floppy
Internal 10/100 NIC
Internal "Sound Blaster" audio card
System Manufacture date: 12/10/96
FCC ID #hwythp5133batc

#1 - Western Digital – configured as the master drive according to the
illustrations on the disk case.
                  Model Caviar 11000
                  2046 cyl
                  16 heads
                  63 spt
                  1055.9 MB
                  MDL: WDAC11000-00H
                  P/N: 99-004222-000
                  CCC: H0 18 NOV 96
```

```
                     DCM: ANAARBL
This disk was recognized by Trinux Linux (9) as hda1.dd
md5: 00d284e2b73ccdbd8313a57bd1e2fafb

#2 – Maxtor – configured as the slave drive according to the illustrations on
the disk case.
                     Model 81750A4
                    4099 MB
                     cyl 3400
                     heads 16
                     sectors 63

This disk was recognized by Trinux Linux (9) as hda2.dd
md5: 628f2390a359da96a6d6127137744d84
```

During the analysis of this system, the same components and configurations were used as during the analysis of the atd binary in part 1 of this paper.

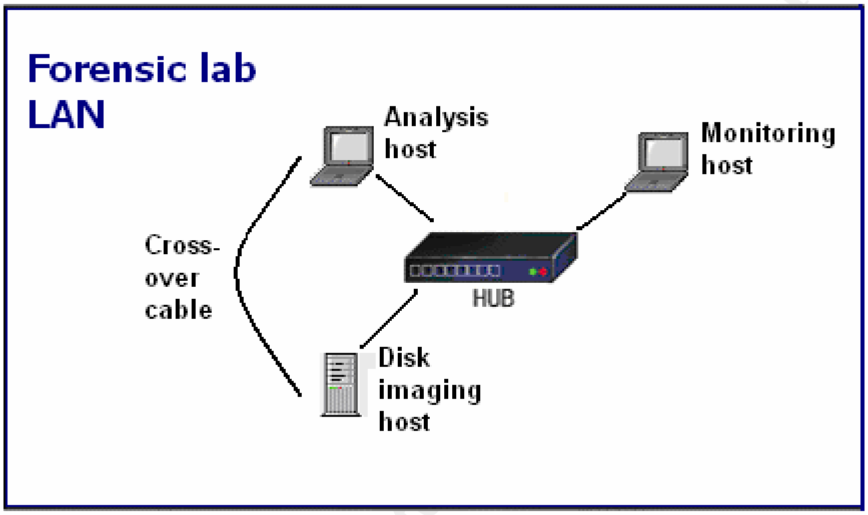


Image Media

First the two hard disks were removed - one at a time - from the host case. As each disk was removed, the following steps were taken:

1. An identifying "label" was written on its case in permanent ink. For this investigation the labels were Disk1 for the IDE master disk and Disk2 for the IDE slave disk.
2. All identifying information present on each disk was recorded into a Hardware Inventory file that is unique for each investigation.
3. The jumper settings were documented. In this case the jumper on the IDE slave disk (Disk2) would later be changed to single/master while being imaged/copied.
4. Each disk was placed in a static free plastic bag, and each bag was clearly marked to identify its contents.
5. The power cables and the ribbon cable connected to each disk was labeled and documented.
6. The physical drive bay that each had been removed from was documented.

The host components, minus the hard disks, were labeled and sealed in plastic bags and stored in a restricted access locked room. Though these components were not expected to be part of any subsequent investigation, they were treated as evidence just as the hard disks were.

This illustrates a point learned during this investigation. The analyst may have a vague idea of the scope and severity of the alleged incident, and those around the analyst usually share their own assumptions, expectations, and speculations. Still, the analyst must work from the start with the assumption that the investigation will go completely through the criminal prosecution process. An analyst never knows when a "routine" investigation might uncover unexpected evidence of much more serious activities, criminal or otherwise.

Next, Disk1 was connected to the ribbon and power cables of the disk image host. Its own hard disk had been disconnected from the bus and power, allowing Disk1 to "take its place." Disk1 was not bolted into the case; it simply laid along side the open workstation case. The BIOS configuration of the disk-imaging host had been confirmed to boot from floppy diskette, CDROM, and fixed disk – in that order.

A Trinux [9] boot diskette was firmly seated in the floppy drive and the system was powered up. Using a Linux boot floppy insured that accidental or unintended access to the hard disk from within the controlled Linux environment would not be able to alter the disk contents. Powering up the processor and properly mounting the hard disk with Trinux would not alter any of the content of the fixed disk being analyzed.

Illustrating the Trinux boot process in detail would not significantly add to the value of this report. All that needs to be mentioned is that during the process, the operator is prompted for any additional "packages" that should be loaded. These packages are compacted archives of common software tools. This provides a great deal of customization in selecting what software the system will contain. It also allows the operator to conserve space (specifically the RAM used to store the contents of the virtual file system) and speed the boot process.

When prompted for additional packages, a diskette containing the diskutl.tgz [9] archive (available at the Trinux web site) was swapped for the boot diskette. This package provides the sfdisk and fdisk utilities.

As Trinux successfully booted from the diskette, one hard disk (hda) was located and found to contain a single NTFS (Net Technologies File System) partition. This implied that this disk was used by a Microsoft Windows NT, or above, operating system.

From the Trinux console, sfdisk was run to scan the system for hard disks, their reported geometry, and the partition table's record(s). The sfdisk tool reported finding one fixed disk it labeled **had,** which contained one NTFS partition it labeled **hda1.**

At this point, it was assumed that none of the proceeding steps had altered the disk or its contents in any way. But the next steps would begin to open the possibility of inadvertent alteration, which would contaminate the disk destroying the credibility of any finds the analysis might produce.

So, before attempting any (other) type of access to the disk/partition, the unique md5 [11] (**M**essage **D**igest version **5** [33]) hash value of this disk had to be calculated and documented. Because the md5 hash value of each thing (a disk, a file, an email, or any digital material) is unique, the analyst can be sure that the disk image file actually being analyzed is identical to the original. This also allows the integrity of the findings to be validated at a later date by repeating them on an identical copy of the original. Comparing the md5 hash values of an original and its copy prove that they are in fact identical.

At this point, the drive was simply a device and had not yet been identified to the system as a file system. In other words, the disk had not yet been mounted so there was no change access in place, only read access.

The command run is illustrated below.

```
# md5sum /dev/hda
00d284e2b73ccdbd8313a57bd1e2fafb       /dev/had
```

The md5sum [11] tool processes the subject file through the md5 algorithm producing a hash value. A hashing algorithm specifically accepts digital material of variable length and produces a value that uniquely identifies the input material such, that if any alteration is made to the material, the resulting hash value will be significantly different. In effect, the hash value is an extremely unique fingerprint of that material. This fingerprinting allows an analyst to demonstrate that the material being analyzed is an identical copy of the original and that it has not been altered from the original. In some cases, the point might be to prove that some alteration has actually occurred.

By md5 hashing the /hda disk rather than a specific partition (/hda1), the hash value would represent the entire disk surface, including the partition table, master boot record, and all other addressable sectors.

Even the file documenting the md5 hash of Disk1 was itself hashed. This is an example of the principal of <u>defense in depth.</u> This layering of safety nets is seen repeatedly in digital forensics. The idea being that, rather than defending against an intruder, you are defending against mistakes, accidents, system failures, and other unexpected events that could damage the integrity of the investigation. This also provides a buffer for responding to unexpected criticism or questions raised later in the process. Why didn’t you have a backup of your backup?

Since all of the information being collected at this point exists only in the virtual Trinux RAM filing system, a blank formatted diskette was mounted (in place of the Trinux boot and package diskettes) and the Disk1.md5 file, along with others, were copied onto it. From there, these files were copied into the investigation’s file folder.

After completing the md5 hash process a byte-for-byte identical copy of the suspect disk was made. This is commonly referred to as *imaging* the disk.

The disk imaging system was connected directly to an Ethernet 10/100 hub (not a switch), as was the analysis host. No other devices were connected to the hub at this time, and no other vectors existed between this lab LAN and any other systems. This LAN was air-gapped, physically and logically isolated from any other network.

On both the disk imaging and analysis hosts, the IP configurations were manually set using the ifconfig tool, since there was (intentionally) no DHCP service available. In the next two examples, ifconfig is given:

- the network interface to be configured (eth0);
- the IP address to be assigned to that interface;
- the netmask for that IP address; and
- finally ifconfig is instructed to set the interface status to "up," or active.

This command was run at the disk-imaging host.

```
# ifconfig eth0 172.24.0.10 netmask 255.255.255.0 up
```

This command was run at the analysis host.

```
# ifconfig eth0 172.24.0.11 netmask 255.255.255.0 up
```

Running ifconfig with no parameters produces a report of the current settings that is used to confirm that the intended action has taken effect.

The ping program was run from each of the two machines at the other to verify successful connectivity. Initially, pings from the disk-imaging host to the analysis host failed due to the strong firewall settings on the analysis host. Since this process was being performed within an air-gapped LAN, the IPTables firewall was disabled.

Now the actual disk imaging process was begun.

The disk imaging host was going to sequentially read the entire hard disk sector by sector and send the data across the LAN to the analysis host, which would spool the incoming data stream to a single disk file. At the disk imaging host, the sender, the following command was issued.

```
# dd /dev/hda bs=512 | nc 172.24.0.10 8841
```

While at the analysis host, the receiver, this command was issued.

```
# nc –l –p 8841 –v –n –w 30 > Disk1.dd.img
```

The first command instructs the host to read directly from the disk device hda in blocks of 512 bytes each. The stream of data being read is piped (the | symbol) directly to the nc [27] (netcat; network concatenation) program that would build a network session to port 8841 on the receiving host (172.24.0.10).

The receiving host reverses the process. A nc listener (-l) is setup on the local TCP port 8841 (-p 8841). Additionally, the nc listener is to verbosely (-v) display its status and not look up any inbound hostnames (-n). Finally, the listener is to continue listening until 30 seconds have lapsed (-w 30) with no inbound data received before terminating. Since nc works with raw packets and streams, it does not have a EOF (end of file) delimiter or flag with which to signal the receiver that the transmission is complete. To compensate for this, the receiver is told to automatically terminate the connection after a period of silence.

As the byte stream flows in from the sender, the receiver directs it to the local disk file Disk1.dd.img.

Initially, parameters such as –w 30 were given on the sender, but that failed. Trial and error showed that this version of the Busybox [(24)] nc program (Busybox provides the functions of many of the programs available to the console) did not support arguments other than host and port. To adjust to this, the commands at the sender are stark, and all of the options were placed on the receiver.

At first there were several false starts. The send-receive process would begin and successfully process several megabytes of data but would always abend (abnormally terminate) at different points in the process.

These failures occurred at very different byte counts and time durations. This implied that the problem was not an unreadable disk sector, but something else. A third host was setup on the hub running tcpdump, a network communications sniffer, in hopes of finding some clue to the problem. After two more failed attempts, the sniffer host had not logged any data that indicated a problem with the hub or the integrity of the hosts' ability to communicate.

Troubleshooting indicated the hub was the source of the problem. In order to keep this investigation on track, and since it would have no impact on the content of the process, a replacement hub was ordered and the two hosts were directly connected with a crossover cable. The crossover cable created a virtual network between these two hosts that would function just as a hub-based network would. Later in the investigation, the replacement hub was put in place and operated as expected.

The send-receive process took over two hours to complete. Since there was no obvious indicator of progress, other than "*the blinky lights*" on the hub, a second terminal was opened on the receiving host and the "ls –o" command was run on the folder that Disk1.dd.img was being spooled into. This gave me an accurate gauge of the transfer's progress, since I could compare the size reported by sfdisk and the amount received in the folder.

Once the disk imaging process was complete, the listener gracefully closed the connection (after waiting 30 seconds with no additional reception as instructed with netcat's "–w 30" parameter).

The byte counts were compared to ensure the entire disk had been successfully imaged. But a byte count is only an indicator and not proof. So, the md5 hash of the Disk1.dd.img file was generated and compared character by character to the original md5 hash of the suspect disk. The hash values matched, proving that the disk image was an identical byte copy of the original.

Once each disk image file was saved to the analysis host’s local disk, the read-only bit was set to help prevent unintentional alteration either by the operator, the tools used for the analysis, or even the local operating system.

The second disk was now imaged.

Both hosts were gracefully powered down and rebooted. The suspect disk, Disk1, was disconnected and returned to its plastic bag.

Disk2 had been configured as the slave drive in the original system. Before connecting it to the disk-imaging host, the proper jumper block on the disk case was moved from the slave position to the master position. This does not alter the disk contents in anyway. Once this disk had been processed, the jumper was moved back to its original position. Fortunately, there was a diagram on the disk case that illustrates the different jumper placement options.

Disk2 was connected to the disk-imaging host just as Disk1 had been. The same environment (systems, hardware) with the exception of swapping Disk1 with Disk2, software, and configuration) were used to image Disk2. As before, rather than using the Ethernet 10/100 hub, a crossover cable was used to connect the sending and receiving hosts in order to transfer the harvested disk image.

Here again, there were problems.

Several attempts were made to harvest an image of Disk2, but input/output errors, reported by dd on the disk-reading host, prematurely terminating the transfer. Unlike the problem with the hub, each attempt consistently failed after approximately 4.9 MB were transferred.

As before, the dd command illustrated below was run at the disk-imaging host.

```
# dd if=/dev/had bs=1024 | nc 127.0.0.37 8841
```

Several different values were tried for the bytes per sector (bs=1024) parameter (512, 1024, 4096...) with no significant improvement.

This implied there was a physical problem with the surface of Disk2 that prevented dd from sequentially reading the entire disk from beginning to end. So, rather than imaging an intact byte copy of the entire hard disk, dd was used to make two images; one an identical copy of the disk from its beginning to the point of damage, and a second that was an identical copy of the disk from just past the point of damage until the end of the disk. These images could not be remounted or analyzed as an intact file system, but could still provide important findings.

The skip=10000 parameter was used with dd to begin reading just past the apparent point of damage on the disk when creating the second image of the disk just past the point of damage and on to the end of the disk.

The above “skip=10000” parameter instructs dd to begin reading 10,000 * 512 (5,120,000) bytes from the beginning of the disk. This effectively skipped over the first 4.9 MB that had already been harvested along with the point of damage on the disk, and begin imaging the remainder of the hard disk.

```
# dd if=/dev/hda bs=512 skip=10000 | nc 127.0.0.37 8841
```

The analysis host receiving this byte stream did so with the following command.

```
# nc –l –p 8841 –v –n –w 30 > Disk2.2.dd.img
```

From this point, the two segments of Disk2 were treated as separate images, rather than a partitioned functioning file system, that logically represent the image of Disk2 minus the 220,000 (5,120,000 - 4,900,000) bytes containing the damaged portion of the disk surface. If this approach did not offer clues, then a more exact slicing would be performed to more exactly isolate the point of damage. The two slices would be concatenate and the damaged sector(s) would be “back filled” in an attempt to provide enough integrity to the image to allow it to function as a file system.

Still treating each portion of disk2 as a file, each portion was hashed and verified to the material captured at the receiver was identical to the material read from the hard disk. In the examples below, the first portion of disk2 was read with dd and piped to the md5sum program. Next the second portion of the disk, beginning 10,000 blocks of 512 bytes each from the beginning of the disk and running to the end, was read and hashed.

```
# dd if=/dev/hda bs=512 | md5sum

# dd if=/dev/hda bs=512 skip=10000 | md5sum
```

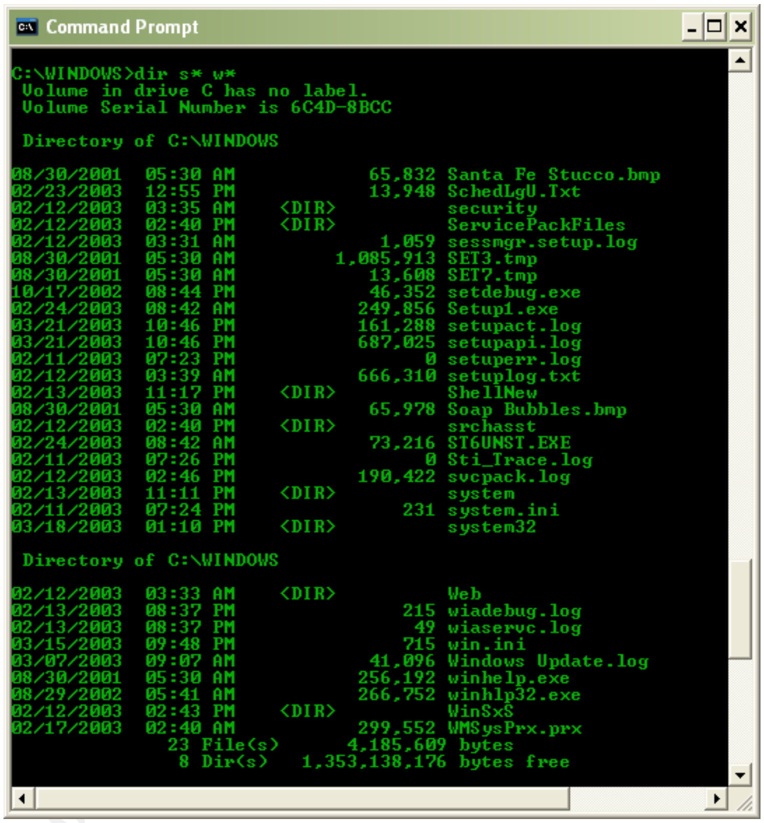
Disk1 Master 1 MB Partition table One fat-16 partition	**Disk2** Slave 4 MB Partition table On NTFS partition
Disk1.dd	Imaged as 2 files Disk2 - first half Disk2.2 - second half

The exercise of creating a disk image helped make clear the concept of a file system as an abstraction layer placed over a logical partition which is itself a abstraction layer placed over the binary contents of a physical disk.

The logical file system

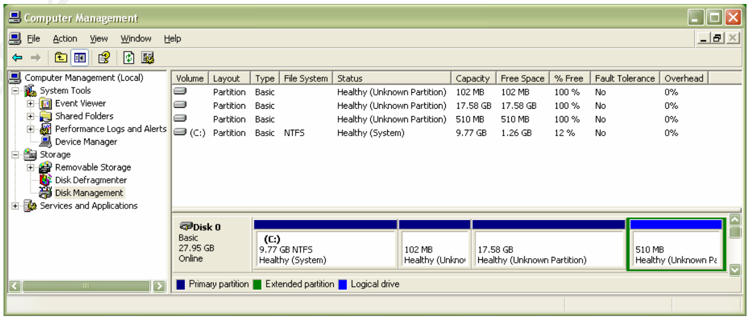
At the point where people and files interact, a logical file system (NTFS, FAT, ext2, ext3…) provides an organized view of the content of each files and its metadata, which includes; date and time stamps, file size, other attributes.

The file system acts like a database management system, presenting the logical data (which is actually organized for efficient access and management by low-level language routines) appear to be organized in the easy to digest (for people) structures of drives, mount points, folders, and files.



```
C:\WINDOWS>dir s* w*
 Volume in drive C has no label.
 Volume Serial Number is 6C4D-8BCC

 Directory of C:\WINDOWS

08/30/2001  05:30 AM            65,832 Santa Fe Stucco.bmp
02/23/2003  12:55 PM            13,948 SchedLgU.Txt
02/12/2003  03:35 AM    <DIR>          security
02/12/2003  02:40 PM    <DIR>          ServicePackFiles
02/12/2003  03:31 AM             1,059 sessmgr.setup.log
08/30/2001  05:30 AM         1,085,913 SET3.tmp
08/30/2001  05:30 AM            13,608 SET7.tmp
10/17/2002  08:44 PM            46,352 setdebug.exe
02/24/2003  08:42 AM           249,856 Setup1.exe
03/21/2003  10:46 PM           161,288 setupact.log
03/21/2003  10:46 PM           687,025 setupapi.log
02/11/2003  07:23 PM                 0 setuperr.log
02/12/2003  03:39 AM           666,310 setuplog.txt
02/13/2003  11:17 PM    <DIR>          ShellNew
08/30/2001  05:30 AM            65,978 Soap Bubbles.bmp
02/12/2003  02:40 PM    <DIR>          srchasst
02/24/2003  08:42 AM            73,216 ST6UNST.EXE
02/11/2003  07:26 PM                 0 Sti_Trace.log
02/12/2003  02:46 PM           190,422 svcpack.log
02/13/2003  11:11 PM    <DIR>          system
02/11/2003  07:24 PM               231 system.ini
03/18/2003  01:10 PM    <DIR>          system32

 Directory of C:\WINDOWS

02/12/2003  03:33 AM    <DIR>          Web
02/13/2003  08:37 PM               215 wiadebug.log
02/13/2003  08:37 PM                49 wiaservc.log
03/15/2003  09:48 PM               715 win.ini
03/07/2003  09:07 AM            41,096 Windows Update.log
08/30/2001  05:30 AM           256,192 winhelp.exe
08/29/2002  05:41 AM           266,752 winhlp32.exe
02/12/2003  02:43 PM    <DIR>          WinSxS
02/17/2003  02:40 AM           299,552 WMSysPrx.prx
              23 File(s)      4,185,609 bytes
               8 Dir(s)   1,353,138,176 bytes free
```

The logical partition



Volume	Layout	Type	File System	Status	Capacity	Free Space	% Free	Fault Tolerance	Overhead
	Partition	Basic		Healthy (Unknown Partition)	102 MB	102 MB	100 %	No	0%
	Partition	Basic		Healthy (Unknown Partition)	17.58 GB	17.58 GB	100 %	No	0%
	Partition	Basic		Healthy (Unknown Partition)	510 MB	510 MB	100 %	No	0%
(C:)	Partition	Basic	NTFS	Healthy (System)	9.77 GB	1.26 GB	12 %	No	0%

Disk 0 Basic 27.95 GB Online	(C:) 9.77 GB NTFS Healthy (System)	102 MB Healthy (Unkno	17.58 GB Healthy (Unknown Partition)	510 MB Healthy (Unknown Pa

▇ Primary partition 🟩 Extended partition 🟦 Logical drive

As the file system presents a digestible view of the organization of files, a logical partition presents a digestible view of the more physical aspects of a hard disk to the file system so a human can better interact with it. This type of "hardware abstraction layer" allows the upper layer (the file system) to focus on the tasks of a file system without having to deal with the differences and variations between hard disk products.

In this illustration, Windows XP reports finding four logical partitions on this one hard disk. The last in the list is the first in the associated graphic. This is the 9.77 GB NTFS partition used by Windows XP as the C: drive.

The other three partitions, which XP does not recognize because it does not provide native (built-in) support for ext2 or other *nix file systems, are the ext2 file system /boot (102 MB), /(logical root 17.58

GB), and /swap (510 MB) partitions used by Linux on this dual-boot system.

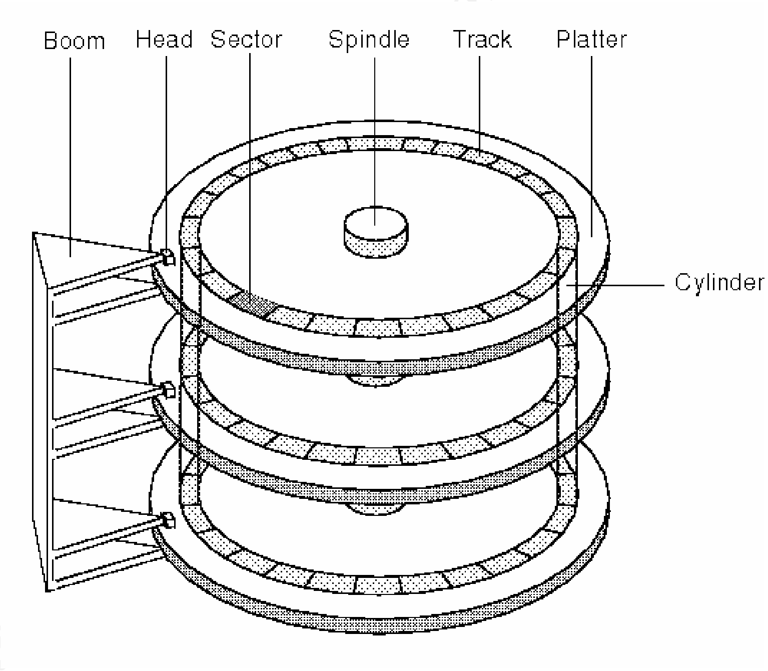
XP has gotten this information from the partition table on this disk. Each physical hard disk has a partition table that an operating system (Linux, Windows, DOS, Solaris…) relies on to describe the logical partitions that the hard disk has been divided into.

The physical disk

Even at this lowest layer, the hard disk itself is managed and accessed through firmware on the Disk Controller Board (DCB).

Usually, this firmware is hardware-specific and resides in chips that are physically integrated into the hard disk chassis (as seen below).

At this layer of abstraction, the physical characteristics of a hard disk are masked from the software executing on the host.



Media Analysis of System

The analysis system

Normally, a desktop system with ample resources would be used for this type of work.

- A desktop system allows for easy access to the internal components, such as the hard drive bus and power cables, that are used to connect a “suspect” hard disk.
- A desktop system is usually kept in an easy to secure location with environmental controls (rather than the trunk of your car).
- Desktop systems and their resources (RAM, disk, SCSI controllers, external tape drives._._.) are comparatively less expensive than the laptop versions.
- Desktop systems are usually more comfortable to work at for extended periods of time due to larger and crisper video displays and more ergonomic input device (keyboard and mouse).

However, due to resource and budget restraints, a laptop host was used for the analysis system.

The system was configured to dual-boot Redhat Linux 7.3 and Windows XP. It contained 256 MB of RAM, an internal 10/100 Ethernet NIC, modem (not used during this project), 3.5 internal floppy drive, and CDRW drive.

During the investigation, frequent backups were made of the files generated during the investigation. These backups were encrypted and burned to CD-ROM.

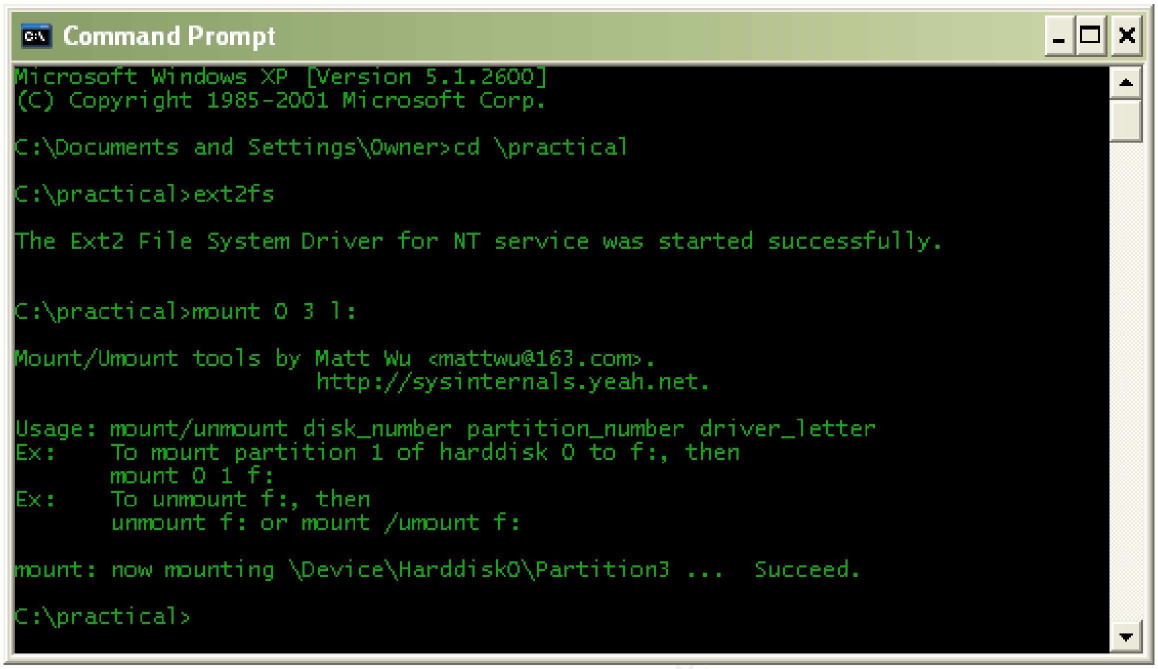
The 20 GB disk was portioned as follows:

1. One 9.77 GB NTFS boot partition for Windows XP
2. One 102 MB boot partition, formatted for the ext2 file system
3. One 17.58 GB logical root (/) partition
4. One 510 MB ext2 /swap partition

Rather than keep duplicate copies of the disk image files on both the Linux and XP partitions, read-only access was established from one operating system to the other's partition.

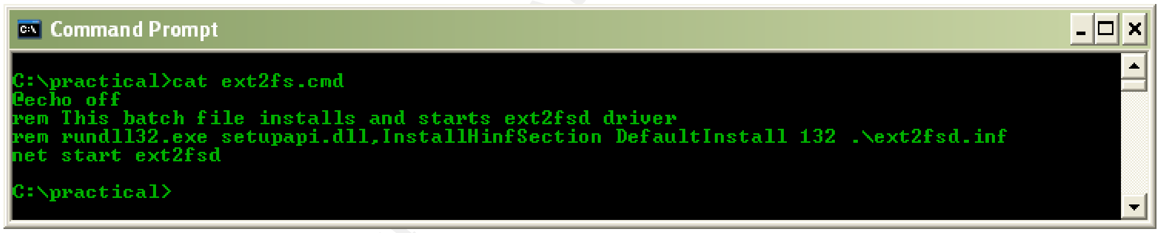
On xp, the ext2fs [3] file system support was installed to provide read-only access from XP to the Linux ext2 file system partitions. As illustrated below, from an XP console, the ext2fs batch file, followed by the mount command, establishes a mount point to the ext2 partitions.

The ext2fs batch file initializes the ext2 file system driver for Windows, and the mount command makes an ext2 file system accessible to Windows as a drive letter. In this case, the mount command is instructed to associate a drive letter "mount point" labeled l:\ with the third partition on the local hard drive 0, which is actually the ext2 root (/) partition on this drive. From this point, the ext2 root partition l: on this disk is available to the Windows operating system as the l: drive.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Owner>cd \practical

C:\practical>ext2fs

The Ext2 File System Driver for NT service was started successfully.


C:\practical>mount 0 3 l:

Mount/Umount tools by Matt Wu <mattwu@163.com>.
                      http://sysinternals.yeah.net.

Usage: mount/unmount disk_number partition_number driver_letter
Ex:    To mount partition 1 of harddisk 0 to f:, then
       mount 0 1 f:
Ex:    To unmount f:, then
       unmount f: or mount /umount f:

mount: now mounting \Device\Harddisk0\Partition3 ...  Succeed.

C:\practical>
```

The ext2fs.bat file (shown below) is used to install and initialize the file system driver for XP. Since the installation need only be run once, the rundll32.exe line is commented out, leaving only the net start line to actually be called from this batch file.



```
C:\practical>cat ext2fs.cmd
@echo off
rem This batch file installs and starts ext2fsd driver
rem rundll32.exe setupapi.dll,InstallHinfSection DefaultInstall 132 .\ext2fsd.inf
net start ext2fsd

C:\practical>
```

Now Windows and its applications have line-of-sight, read-only access to the contents of the ext2 partitions.

Establishing read-only access for Linux to the ntfs partition is simply a matter of mounting that partition, /dev/hda1 in this case.

```
# mount /dev/hda1 /mnt/ntfs -r -o ro,noatime,nodev,noexec

# mount
/dev/hda3 on / type ext2 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/hda2 on /boot type ext2 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/dev/hda1 on /mnt/ntfs type ntfs (ro,noexec,nodev,noatime)
```

Using a laptop allowed for analysis work to be performed during non-office hours and at locations other than the office – something normally a violation of accepted forensic practices. An analyst must maintain the integrity of the chain-of-evidence that includes, not only the original material (in this case the physical disks), but also for the working copies on which the actually analysis is performed. Moving the material being analyzed on an easy to lose/steal laptop computer is simply unacceptable.

When a forensic investigation begins, you do not know what you might find, regardless of your expectations or the initial details of the situation. Your task may simply be to recover a file accidentally deleted by a trusted employee, only to find that the system contains confidential or illegal material the employee has no knowledge of. Any benign situation has the possibility of unexpectedly becoming a criminal matter. So, every investigation must be performed from the start as if it was a serious criminal matter that would be fully scrutinized in a court of law, or worse, the court of public opinion.

In this case, that principle was disregarded, since the outcome was already known. Still, forensic analysis best practices needed to be acknowledged and followed.

So the following conditions were observed.

1. Access control – With no exceptions, only the analyst was allowed physical or logical access to the physical and logical components of the investigation.
2. Encryption – All of the material (files) being analyzed were kept encrypted when not actually being analyzed. The encryption was performed using gpg [25] with a 2048-bit key created specifically for this project. At the end of each session, the unencrypted material was destructively overwritten with the wipe [26] utility.
3. Physical security – when not in use, all components were kept in a locked desk drawer, in a locked office.
4. Encrypted copies of the material being analyzed, working notes, and the findings were backed up to removable media (Iomega [27] 100 MB zip) and CD-ROM.

The characteristics of each of the disk image files were then documented.

Notice that the read-only bit is set for each disk image file. This is another layer in the *defense in depth* intended to protect the integrity of the material being analyzed and any subsequent findings.

```
# ls *.dd –o
-r--------   1 root    1055899648 Mar  6 14:44 disk1.dd
-r--------   1 root    2148302848 Mar 6 10:26 disk2.2.dd
-r--------   1 root     4934656 Mar  6 15:29 disk2.dd

# md5 *.dd
00d284e2b73ccdbd8313a57bd1e2fafb      disk1.dd
01ce1e7a82928a960133331f68f4fff3      disk2.2.dd
2499a815c217055468da0941d823de6e      disk2.dd
```

Because the disk image files are just that, identical copies of functional disks supporting functional file systems, the sfdisk tool can be used to interrogate an image file as if it were a physical disk.

Below, sfdisk is used with the –l (list all discoverable disks and their partition tables) parameter to document each image. These reports are also compared to sfdisk's reports of the original disks.

Of particular interest to the analyst, the partition table can help identify each logical volume, along with any unallocated portions of the disk. This maps out the file systems to be analyzed and the "raw" portions of the disk that need to be analyzed for material that may not be associated with a file system. This is done by correlating the aggregate block sizes with the total disk size and noting any unallocated gaps between partitions.

The reports for the Disk2 are unusual, but fully accounted for by the fact these image files are actually the two slices of a single disk.

Describe each tool used to examine the system and why that tool was used.

There were many tools used in this analysis, each selected for the quality and usability of its results. Usability generally refers to how well the output presents its findings, or lends itself to further processing with other tools. For example, grep is very useable because its output can easily be passed to other text tools for additional processing or formatting. Usability also means how well the tool can be used from a command line or scripted.

While there are some very useful GUI (Graphical User Interface) tools available, the fluid nature of the analysis process and the necessity for granular control of the environment calls for more flexibility of than a GUI generally permits.

A specific exception to this is Brian Carrier's Autopsy [(50)]. Unlike many GUI/browser-based tools that generate lots of data and reports, Autopsy enhances the analyst's granular access to data, while providing results presented in a clear and accurate context.

Show how your tools did not modify the evidence when performing your examination.

For the most part, confidence in the integrity of the tools used in this investigation was based on the observations and testing of others.

However, each tool was acquired from a trusted source and the md5 hash of each was validated from a second trusted source. In addition, many of these tools are available for both the Windows and Linux operating systems. When possible, the findings of a tool in one operating system is compared and validated using the same tool in the other operating system (using the same data of course).

At several points during the analysis, the MAC times and md5 hash values of the suspect disk image files were compared to the original values to verify that the material being analyzed had not been altered by the analysis process.

1.Examine file system for modification to operating system software or configuration

To begin analysis of the file system on Disk1, the file [16] utility was used to help identify the file types. Trinux had reported finding specific file systems, but those reports need to be confirmed by a second tool: in this case, the file tool.

The report produced by the file program below illustrates that disk1 is recognized to have a valid boot sector. Disk2 also contains a valid boot sector, though its logical file system will be unusable due to splitting of disk2 into two parts. Because Disk2.2 is the second part of a partitioned file system and does not contain header information such as a boot sector, it is reported to be data; accessible but unable to be classified.

```
# file *.dd

disk1.dd:    x86 boot sector
disk2.dd:    x86 boot sector, FAT (32 bit)
disk2.2.dd:  data
```

The fsstat [42] (**f**ile **s**ystem **stat**istics) tool performed a deeper interrogation of each disk image. As noted below, disk1.dd contains a FAT16 file system. This report provides a few clues worth noting. The OEM is MSWIN4.0 implying Win NT 4.0. The volume ID is another piece of correlating information that identifies this unique instance of the file system. The geometry reported in the CONTENT-DATA INFORMATION section helps map out the disk image before digging into it with a sector editor.

```
# fsstat -f fat32 disk2.dd

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT
OEM: MSWIN4.1
Volume ID: 402135417
Volume Label: WIN-95
File System Type: FAT32

META-DATA INFORMATION
--------------------------------------------
Range: 2 - 134424578
Root Directory: 2

CONTENT-DATA INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 4096
Sector of First Cluster: 16458
Total Sector Range: 0 - 8417995
FAT 0 Range: 32 - 8244
FAT 1 Range: 8245 - 16457
Data Area Sector Range: 16458 – 8417995
```

Since Disk2 had been sliced into two pieces, the second half was an incomplete portion of a file system, and not usable by file system utilities, since most of the file system management data is at its beginning. The first portion of the disk contained that management data, which could be successfully queried, but appears to be corrupted.

In the illustration of fsstat below, you can see that several items seem to expect text data, but actually contain binary material. However, even though the logical file system may be damaged and inaccessible, the raw (disassociate) data it contains *is* assessable with tools like strings, grep, and hexdump.

```
# fsstat -f fat16 disk2.2.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT
OEM: ?м|??P
Volume ID: 1962949760
Volume Label: ????u???
File System Type (super block): µ?L????

META-DATA INFORMATION
--------------------------------------------
Range: 2 - 3355604354
Root Directory: 9
```

Next, fsstat reports that an ntfs file system has been identified on this partition. An interesting option that was not mentioned earlier is to include the parameter show_sys_files=TRUE to the option list in the mount program. This option reveals the many files in the C:\root, and elsewhere that are normally locked by the Win32 operating system and unavailable to an operator while Win32 is running.

```
# fsstat -f ntfs disk1.dd

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: NTFS
Volume Serial Number: EC6721A2EC6721A2
Version: Windows NT

META-DATA INFORMATION
--------------------------------------------
Range: 0 - 20492
Root Directory: 5

CONTENT-DATA INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 512
Total Cluster Range: 0 - 2062303

Attribute Defs (MFT Entry: 4)
$STANDARD_INFORMATION: 16
$ATTRIBUTE_LIST: 32
$FILE_NAME: 48
$VOLUME_VERSION: 64
$SECURITY_DESCRIPTOR: 80
$VOLUME_NAME: 96
```

```
$VOLUME_INFORMATION: 112
$DATA: 128
$INDEX_ROOT: 144
$INDEX_ALLOCATION: 160
$BITMAP: 176
$SYMBOLIC_LINK: 192
$EA_INFORMATION: 208
$EA: 224
```

Before mounting the disk images and accessing their contents through their file systems, the first sector (512 bytes) was displayed. Both seem to support the previous findings. The hexdump tool compliments dd when it comes to working with binary files. Hexdump provides a granular view of the file, helping to identify the specific byte(s) you want to parse out with dd.

```
# hexdump –C -n 512 disk2.dd

00000000  eb 5b 90 4e 54 46 53 20  20 20 20 00 02 01 00 00  |.[.NTFS    .....|
00000010  00 00 00 00 00 f8 00 00  3f 00 20 00 3f 00 00 00  |........?. .?...|
00000020  00 00 00 00 80 00 80 00  e0 77 1f 00 00 00 00 00  |.........w......|
00000030  d9 1b 02 00 00 00 00 00  f0 bb 0f 00 00 00 00 00  |................|
00000040  02 00 00 00 08 00 00 00  a2 21 67 ec 36 67 ec 84  |.........!g.6g..|
```

Now that the disk image files have been documented and many of their attributes (size, file system type) are now known, the image files can be mounted for direct review.

In the example below, two directories (/mnt/vfat and /mnt/ntfs) were first created to act as the mount points for these disk images.

Because disk2 is an ntfs file system, the show_sys_files=true option is included in its mounting. This will make visible many otherwise hidden files used by the OS to manage the file system.

The third mount command, having no parameters, report on all currently mounted mount-points. This is used to confirm that the intended settings are in fact in place and operating. This is to make sure that the intended read-only access is in force before proceeding.

```
# mount disk1.dd /mnt/vfat t vfat -r –o ro,loop,noatime,noexec,nodev
# mount disk2.dd /mnt/ntfs t ntfs -r -o ro,loop,noatime,noexec,nodev,show_sys_files=true

# mount
/dev/hda3 on / type ext2 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/hda2 on /boot type ext2 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/dev/hda1 on /mnt/ntfs type ntfs (ro,noexec,nodev,noatime)
/root/morgue/disk1.dd on /mnt/vfat type vfat (ro,noexec,nodev,noatime,loop=/dev/loop0)
/root/morgue/disk2.dd on /mnt/ntfs type ntfs
(ro,noexec,nodev,noatime,loop=/dev/loop0,show_sys_files=true)
```

Later, from XP, disk1.dd file was examined using Hex Workshop [(6)] to inspect the disk image file directly. This confirmed the findings of the file utility.

In determining the OS version, the initial observations clearly indicated Windows NT. Remember that earlier, the Linux mount tool identified the file systems as fat and NTFS, indicating NT or higher as the OS.

- The file systems were fat on the master disk and NTFS on the slave.
- The master disk included the NTLDR, boot.ini, and pagefile.sys files, along with a WINNT folder. All of the expected files and folder for the NT OS were present on disk1, indicating that it was, in fact, an NT boot and system disk (it contained the OS and the partition table, indicating it was the active partition).

In order to inspect the file system contents on Disk1, the analysis host was booted into Linux so that it could mounted using a loopback device.

The parameters used when mounting a partition are critical. An analyst must adhere to the forensics prime directive not to alter the subject material in any way or degree. However, the Heisenberg Uncertainty Principle can loosely be applied here. Paraphrased, the Heisenberg Uncertainty Principal states, “The simple action of observing the subject will, in fact, alter the subject.” In the case of digital forensics, simply querying if a file exists can alter the files metadata such as the time last accessed stamp. In this case, the atime (last **a**ccess time) can be critical in building a timeline of activity on the suspect system. Considering that this is a Windows system, there is always the chance that the OS will “do us the favor” of upgrading or refreshing file components without first notifying us. An example would be Windows feather that automatically replaces certain missing OS files. To observe this, rename c:\winnt\system32\user.exe to user.exe.old. An immediate dir will show that the file has been renamed as expected. However, after a few moments, another dir will report that the user.exe file has reappeared.

So to protect the original state of the material being imaged or analyzed, it is mounted with implicit controls to prevent any alteration to the file system.
The parameters used in this example are interpreted as:

1. **-t ntfs** informs mount of the **t**ype of file system to be mounted.
2. **o** is followed by a comma-delimited list of **o**ptions.
3. **ro** is the **r**ead-**o**nly option, instructing the file system interface to allow no changes to be made to the file system.
4. **noatime** instructs (redundantly) the file system interface to allow no updating of the access time metadata associated with each file in the partition. Since simply listing the existence of a file can alter its date time last accessed meta-data.
5. **loop** the loop device is used to create a layer of abstraction between hardware and software. In fact, this is so abstract that there is no hardware. Any software that attempts to access this loop device must do so as if it were an actual physical device, while the device itself acts as if it were hardware.
6. **nodev** declares that no code libraries stored on this mount point are to be used by programs in execution.

7. **noexec** is the last option in this example and instructs the file system interface to allow no executables stored on this mount point to be executed for any reason.
8. **show_sys_files=yes** will display many system files that are not usually visible, and that are used by the OS to manage the file system.

At this point, the investigation begins to shift from the collection of potential evidentiary material the analytical sifting of this material in search of clues and evidence. Since this system being analyzed is Win32, throughout the searching tasks information from The Riddler's paper Microsoft's Really Hidden Files: A New Look at Forensics (46) is used to help locate files of interest.

With the logical file system now accessible, an inventory of its files can be created. This list will not include any deleted files or file fragments, only the files considered viable by the file system. This file.inventory file will serve several purposes.

First this file inventory will document each file and its attributes as they were went when received for analysis.

Second, rather than search through the raw file system itself, which requires it be mounted, the grep tool can be used to search through this one file much faster. One disadvantage is the disk space required to duplicate this information. While the file inventory may not create a problem, keeping a copy of the strings output, or the unallocated sectors can more then double the size of the original disk image file. Then space is a problem; just fall back on a full stringing or greping each time it is needed.

Finally, the atime stamps that will be included in this report can be used to collaborate the timeline that will be generated later with other tools.

The following ls command is used to build the file.inventory file.

The first parameter instructs ls to create a detailed, one-file-per-line list that omits the group owner of each file, making each line a bit shorter.

The –-full-time parameter instructs ls to include the entire timestamp of each file, rather than just a simple date. The –-time=atime parameter specifies the **a**ccess time, rather than the modified or changed timestamps, is to be displayed in the reported.

Finally, the –R parameter tells ls to process each file in each folder **r**ecursively. This is accomplished by drilling down through the entire directory structure beginning at the current directory point, which, in this case, is the logical volume's root. In other words, report on every file in the file system.

To complete the process, the output of the ls command is redirected from the screen to the disk1.file.inventory file.

```
# ls –o –-full-time –-time=atime –R > disk1.file.inventory
# wc disk1.file.inventory
 48349  351465 3656521 disk1.file.inventory
```

The ls command has identified and documented 48,349 individual files. Below the WINNT\repair folders illustrate the detail of information recorded about each file.

```
./WINNT/repair:
total 324
-r--------    1 root         438 1999-07-19 09:56:12 -0400 autoexec.nt
-r--------    1 root        2510 1999-07-19 09:56:12 -0400 config.nt
-r--------    1 root       18283 1999-06-25 16:44:03 -0400 default._
-r--------    1 root       16606 1999-06-25 16:44:08 -0400 ntuser.da_
-r--------    1 root        3336 1999-06-25 16:44:05 -0400 sam._
-r--------    1 root        6991 1999-06-25 16:44:04 -0400 security._
-r--------    1 root       50726 2001-09-22 12:42:14 -0400 setup.log
-r--------    1 root      130453 1999-06-25 13:56:42 -0400 software._
-r--------    1 root       93036 1999-06-25 16:43:59 -0400 system.
```

The file inventory is now used to identify files of potential interest. The contents of the filehunt file are cat'ed (concatenated, displayed) to the screen, showing the filename fragments that will be fed to grep. Grep will search through the inventory file, noting every instance of a filename that matches these fragments.

Rather than manually browse through the entire file system, grep is used to examine the name of each file in the file system, creating a list of the files that are of the most interest.

Note that for convenience, the disk1.file.inventory was edited (search and replace) to remove the ".000000000" string, which was at the end of each timestamp. This simply shortened the lines for easier viewing.

The strings in filehunt are formatted specifically as input for the grep tool. The "\." Portion is interpreted as a literal period (.). Normally, the period can have special meaning as a meta character (a character that implies information about information), but by preceding the period with a backslash, grep will clearly understand the the period is to be treated as part of the literal string rather than a meta-character.

The dollar sign ($) at the end of each string signifies the string being searched for must occur at the end of a logical line rather than the beginning or end of the line. In other words, \.pst$ says find all lines that end with the string literal .pst.

Note that not all lines are identical in regular expression syntax.

- The .tgz string does not require that it be at the very end of the line, this allows files like file.tgz.gz to be identified.
- The sam$ string does not require the preceding period as the others do,
- And the passwd$ and shadow$ files can be preceded by anything.

Finally, the wc tools is used to provide a total. Often, the output of any number of piped programs is run through wc at the end simply to give a numeric indication of the aggregate results; was anything found, and if so, how many?

Here, grep identifies 4551 instances of one of these strings being matched in the file list.

```
C:\practical>cat filehunt
\.pst$
\.mbx$
\.dbx$
\.zip$
\.tar$
\.tgz
\.pwl$
\.pgp$
\.gpg$
\.chk$
sam$
\.bak$
\.tmp$
\.log$
\.mdb$
\.xls$
\.doc$
passwd$
shadow$
\\Windows\\sysbckup
C:\practical>grep -i -f filehunt disk1.file.inventory > files.of.interest

C:\practical>wc files.of.interest
  4984   39892  403715 files.of.interest
```

At this point, the contents of the /mnt/disk1 loopback mount-point are copied to a regular folder on this ext2 drive. The copy will be accessible to XP through the ext2fs driver for analysis performed with Win32 tools.

A forensic analysis is continually presented with much more data than can manually be reviewed. One of the goals throughout this investigation was to identify not just the objects of interest such as log files or password files, but to also cull out those hits that are most likely to be relevant to the analysis.

For example, 100 log files might be found, but only one may be pertinent. The trick is to put all of the easy to gather information (the lists of files, lists of strings, or lists of processes) into context so that items with the most potential return on the investment of time and effort can be identified early in the process.

While this is much more efficient than searching through a file system interface, there are still over 4,000 hits to review. But this can be pruned back more, eliminating the background noise.

The distribution of file types may point to some specific files that warrant attention first. So using the grep tool, now with its –c for count parameter, I created a simple breakdown of the type of files just identified.

By greping with the –c (count) parameter, a quick-and-dirty distribution of the type of files of interest can be built. Then, after a quick review of these findings, the .pst (Outlook email storage) files appear to be a good place to focus.

```
C:\ > grep –i –c “\.pst$” files.of.interest
2

C:\ > grep –i –c “\.zip$” files.of.interest
20

C:\ > grep –i –c “sam$” files.of.interest
10

C:\ > grep –i –c “\.bak$” files.of.interest
5

C:\ > grep –i –c “\.tmp$” files.of.interest
4448

C:\ > grep –i –c “\.log$” files.of.interest
36

C:\ > grep –i –c “\.mdb$” files.of.interest
2

C:\ > grep –i –c “\.xls$” files.of.interest
12

C:\ > grep –i –c “\.doc$” files.of.interest
16
```

This helps to focus on the files that are more likely to be of forensic value. There are likely others, not to mention those files and file fragments that reside in unallocated sectors, but this provides a practical place to start.

Using grep with the –B 5 argument, the 5 lines immediately locate the archive.pst file – the first of two .pst files. The –B 5 argument instructs grep to additionally display the 5 lines just prior to each “hit.” By manipulating the –B and –A (lines after the hit) the line can be put into context; specifically, the folder in which it resides.

According to the timeline shown later in this paper, 2001-12-03 14:47 is when the system was shutdown for the last time.

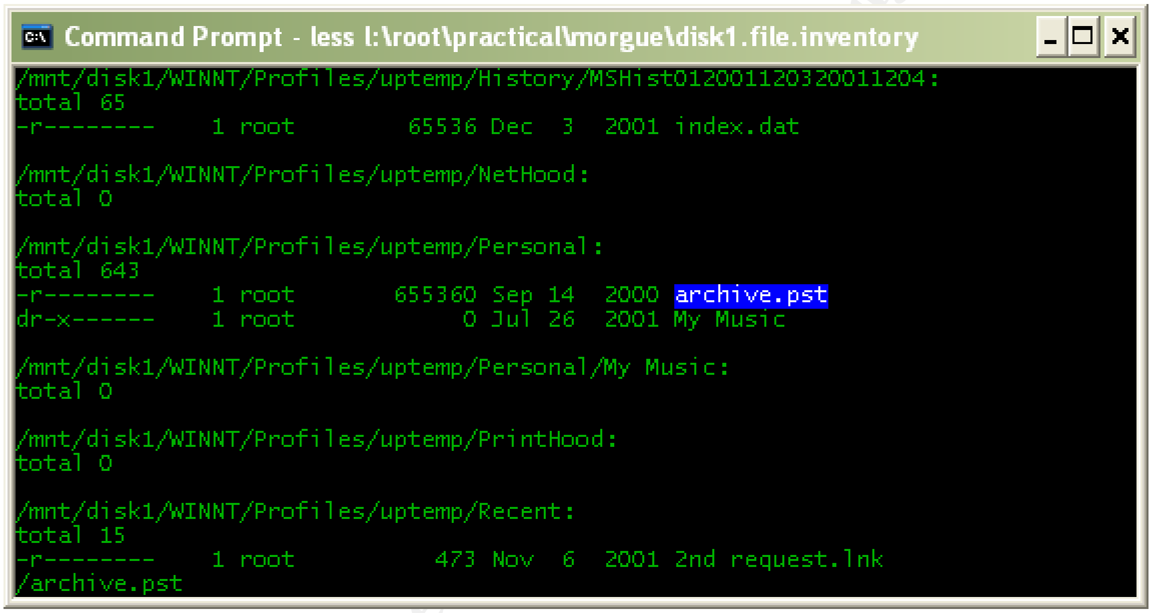
```
C:\practical>grep -i "\.pst" disk1.file.inventory
-r--------    1 root       271360 2001-12-03 14:46:35 -0500 outlook.pst
-r--------    1 root       655360 2001-08-15 15:40:13 -0400 archive.pst
```

The outlook.pst requires more than 5 previous lines to discover its folder. After some trial and error, piping a large (overkill) number of

-B lines into the less utility made it easy to make grep's results useful.

```
C:\PRACTICAL> grep -I outlook.pst -B 200
l:\root\practical\morgue\disk1.file.inventory | less
```

Actually, using the "less" program and its internal search feature, these file names could have been found quicker. An effective analyst has a through knowledge of the available toolset and makes imaginative use of the tools both alone and in combination. This is the tinker-toy principle. Any kid can play with those colored sticks and round wheels with all the holes, but it takes a real engineer to combine all those little, fixed-function pieces into complex, yet stable, systems.

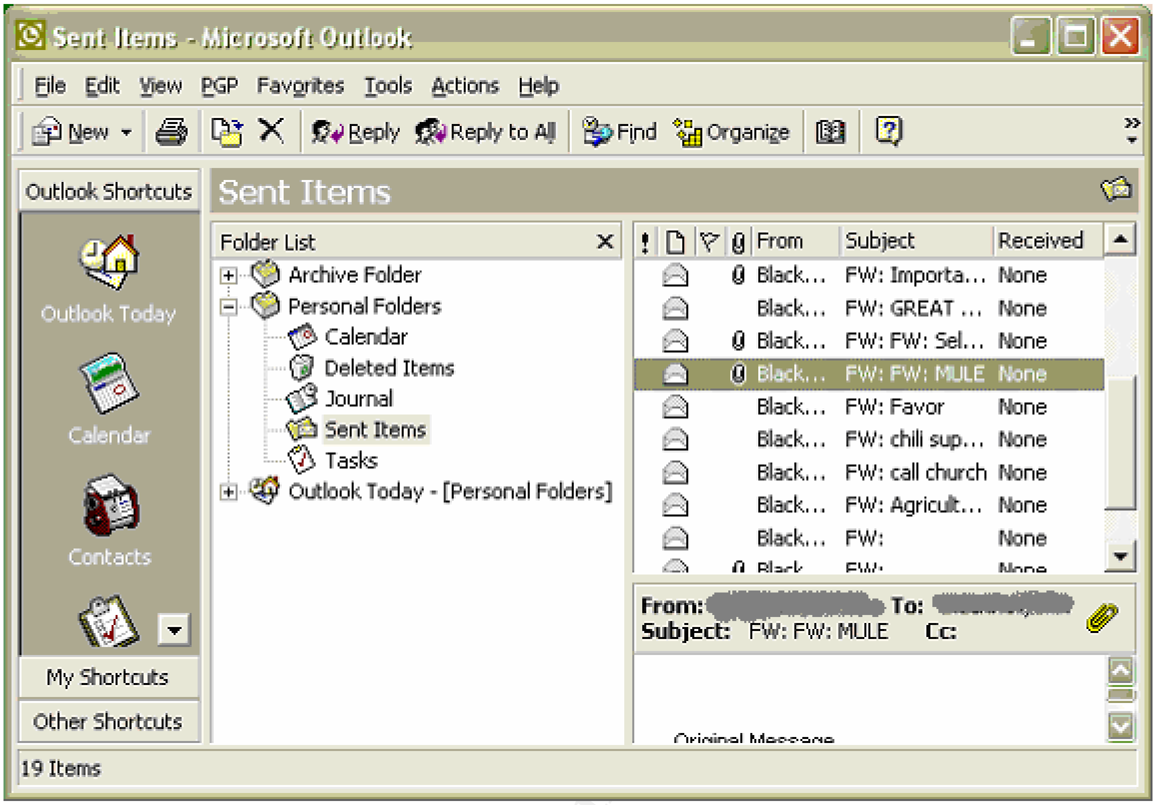

Command Prompt - less l:\root\practical\morgue\disk1.file.inventory

```
/mnt/disk1/WINNT/Profiles/uptemp/History/MSHist012001120320011204:
total 65
-r--------    1 root        65536 Dec  3  2001 index.dat

/mnt/disk1/WINNT/Profiles/uptemp/NetHood:
total 0

/mnt/disk1/WINNT/Profiles/uptemp/Personal:
total 643
-r--------    1 root       655360 Sep 14  2000 archive.pst
dr-x------    1 root            0 Jul 26  2001 My Music

/mnt/disk1/WINNT/Profiles/uptemp/Personal/My Music:
total 0

/mnt/disk1/WINNT/Profiles/uptemp/PrintHood:
total 0

/mnt/disk1/WINNT/Profiles/uptemp/Recent:
total 15
-r--------    1 root          473 Nov  6  2001 2nd request.lnk
/archive.pst
```

The next step is to open each .pst in the Outlook client for direct review. First, each .pst file must be copied out of the read-only loopback device onto a read-write file system, so that the Outlook client program will accept the file. Each .pst is also hashed with md5 to help demonstrate the integrity of the copy being analyzed.

At that point, the .pst files can be opened using a Microsoft Outlook client program, which will act as a front-end to the proprietary file format of the .pst files.

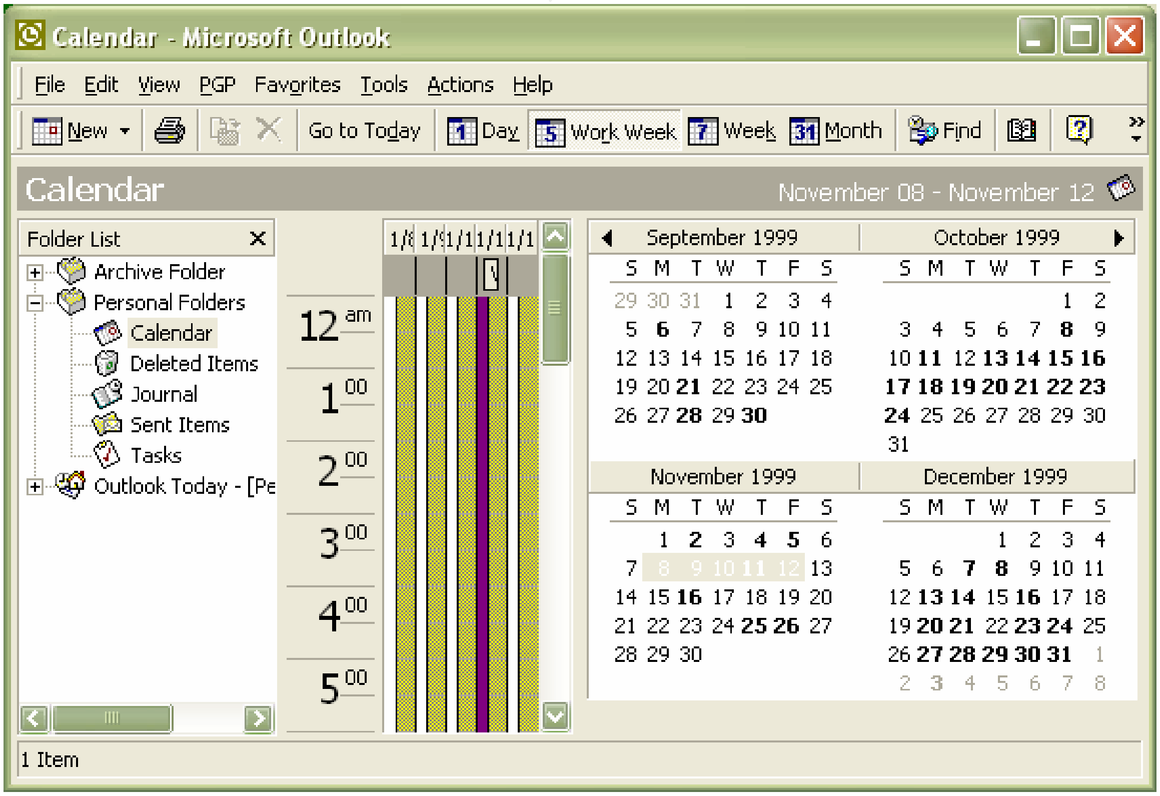
Here we see begin to see the contents of this .pst email archive file. This information helps to identify the people involved with this system, their behavior in using this system, and the names of others who may provide additional evidence or insight.



From	Subject	Received
Black…	FW: Importa…	None
Black…	FW: GREAT …	None
Black…	FW: FW: Sel…	None
Black…	FW: FW: MULE	None
Black…	FW: Favor	None
Black…	FW: chili sup…	None
Black…	FW: call church	None
Black…	FW: Agricult…	None
Black…	FW:	None
Black…	FW:	None

From: [redacted] To: [redacted]
Subject: FW: FW: MULE Cc:

Original Message


Calendar - Microsoft Outlook
File Edit View PGP Favorites Tools Actions Help
New Go to Today 1 Day 5 Work Week 7 Week 31 Month Find
Calendar
November 08 - November 12
Folder List
Archive Folder
Personal Folders
Calendar
Deleted Items
Journal
Sent Items
Tasks
Outlook Today - [Pe
12 am
1 00
2 00
3 00
4 00
5 00
1 Item

September 1999

S	M	T	W	T	F	S
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

October 1999

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

November 1999

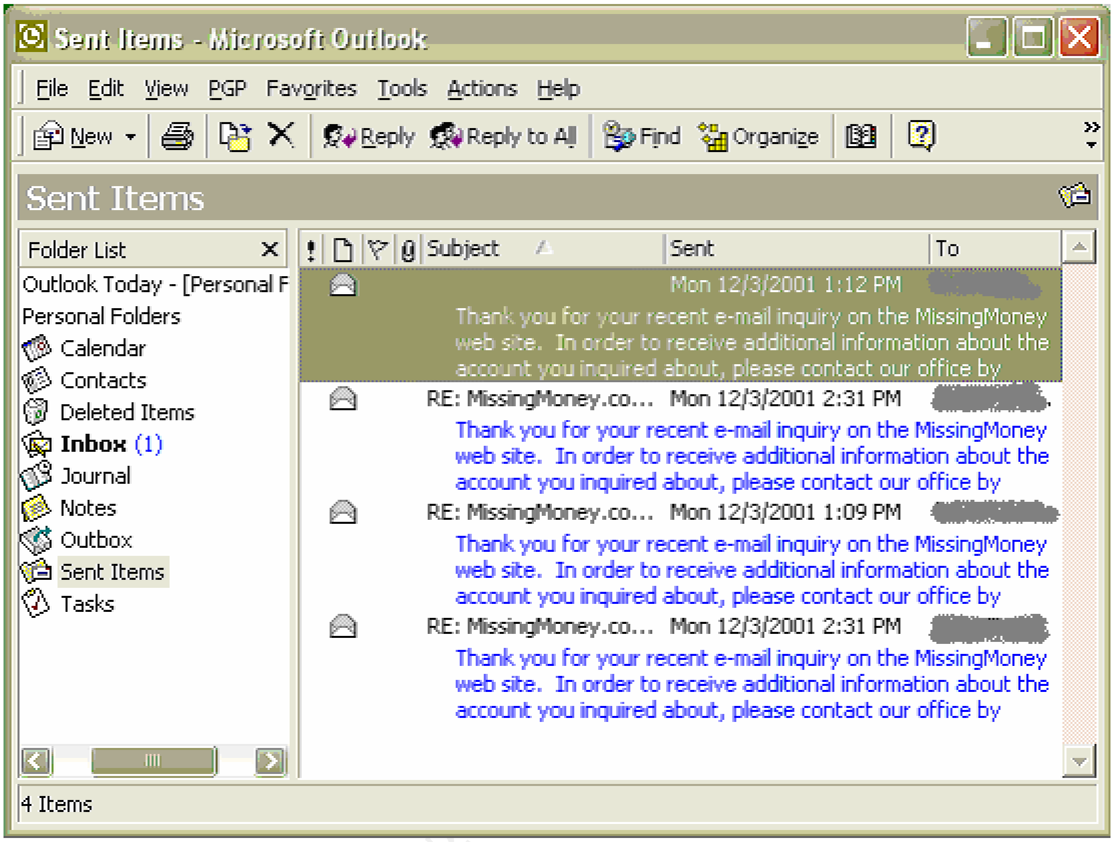
S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

December 1999

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Below the sent items folder in the outlook.pst file can be accessed.

Note that according to the time line, these messages are dated as being sent just before the system was shutdown for the last time.



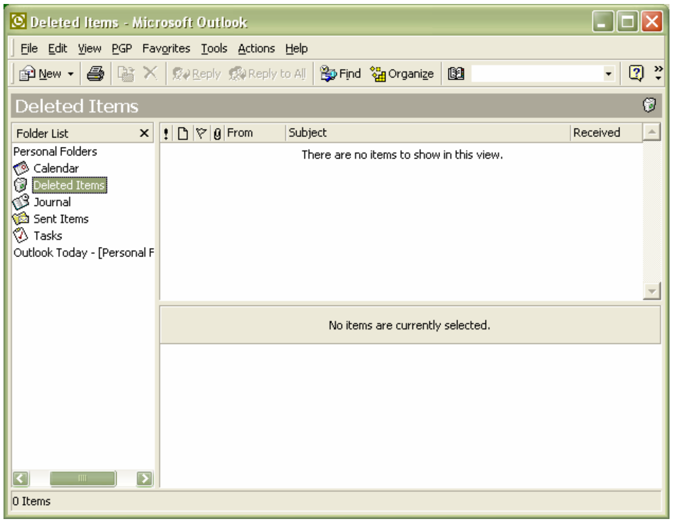
Subject	Sent	To
	Mon 12/3/2001 1:12 PM	[redacted]
Thank you for your recent e-mail inquiry on the MissingMoney web site. In order to receive additional information about the account you inquired about, please contact our office by		
RE: MissingMoney.co...	Mon 12/3/2001 2:31 PM	[redacted]
Thank you for your recent e-mail inquiry on the MissingMoney web site. In order to receive additional information about the account you inquired about, please contact our office by		
RE: MissingMoney.co...	Mon 12/3/2001 1:09 PM	[redacted]
Thank you for your recent e-mail inquiry on the MissingMoney web site. In order to receive additional information about the account you inquired about, please contact our office by		
RE: MissingMoney.co...	Mon 12/3/2001 2:31 PM	[redacted]
Thank you for your recent e-mail inquiry on the MissingMoney web site. In order to receive additional information about the account you inquired about, please contact our office by		

4 Items

This Outlook archive.pst file reported that it contained no *Deleted* email items. However, a new (new to me) technique described by HTCIA International in their September 2002 issue [(29)] raises the possibility that items that have been deleted from the *Deleted Items* folder can be recovered, even after the *Deleted Items* folder has been emptied.

When first viewing the *Deleted Items* folder, it contained no items.

In fact, each folder in this archive was examined. There were only a few emails in the inbox and a few calendar events.



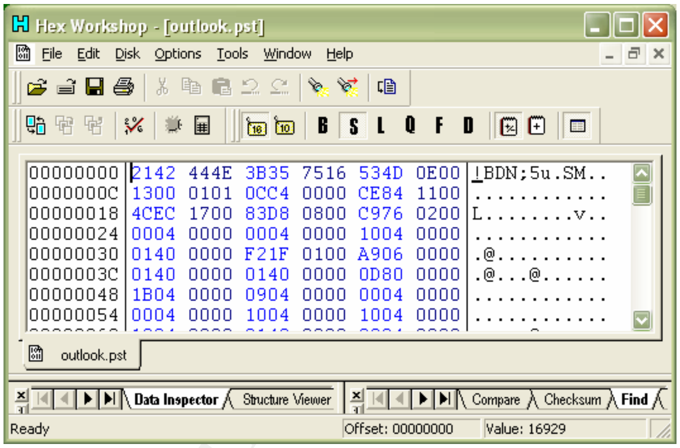
There are no items to show in this view.

No items are currently selected.

0 Items

The first step is to open the .pst file in a hex editor so that you have byte level editing capabilities. Most editors automatically "handle" non-printable characters on the assumption that the file content should be made as digestible as reasonable. A hex editor makes no such assumption, displaying the hexadecimal value of each byte, its printable character, and a dot (.) for non-printable characters.

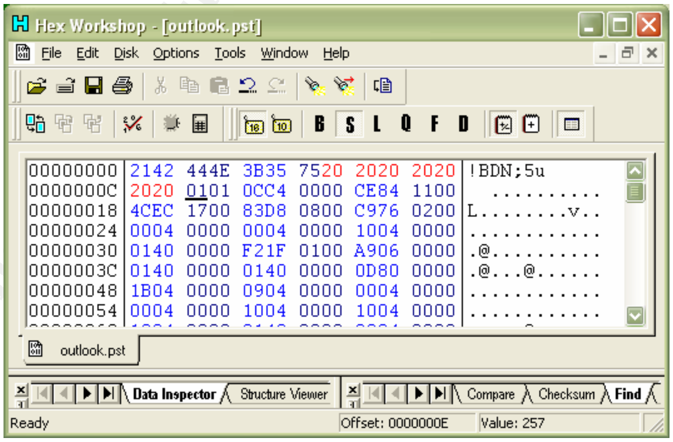
Here you can see the original hex content beginning at sector offset 0000:0000



```
00000000 2142 444E 3B35 7516 534D 0E00 !BDN;5u.SM..
0000000C 1300 0101 0CC4 0000 CE84 1100 ............
00000018 4CEC 1700 83D8 0800 C976 0200 L........v..
00000024 0004 0000 0004 0000 1004 0000 ............
00000030 0140 0000 F21F 0100 A906 0000 .@..........
0000003C 0140 0000 0140 0000 0D80 0000 .@...@......
00000048 1B04 0000 0904 0000 0004 0000 ............
00000054 0004 0000 1004 0000 1004 0000 ............
```

Ready   Offset: 00000000   Value: 16929

Using a hex editor, change bytes 7-13 to ASCII spaces (0x20) and save the changes. In this example, the hex values that were changed are shown in red.

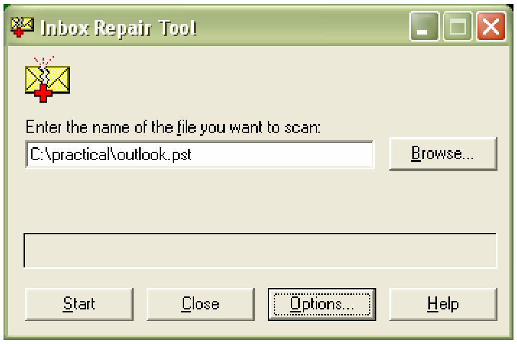


```
00000000 2142 444E 3B35 7520 2020 2020 !BDN;5u     
0000000C 2020 0101 0CC4 0000 CE84 1100   ..........
00000018 4CEC 1700 83D8 0800 C976 0200 L........v..
00000024 0004 0000 0004 0000 1004 0000 ............
00000030 0140 0000 F21F 0100 A906 0000 .@..........
0000003C 0140 0000 0140 0000 0D80 0000 .@...@......
00000048 1B04 0000 0904 0000 0004 0000 ............
00000054 0004 0000 1004 0000 1004 0000 ............
```

Ready   Offset: 0000000E   Value: 257

This small change effectively corrupts the .pst as far as the proprietary file format is concerned. While no damage is done to the email or other items within this archive, the structure itself is damaged as far as the client program (outlook.exe) is concerned, though easily "fixed" to Outlook's satisfaction.

In fact, the next step is to use the scanpst.exe tool to repair the artificially corrupted .pst file.

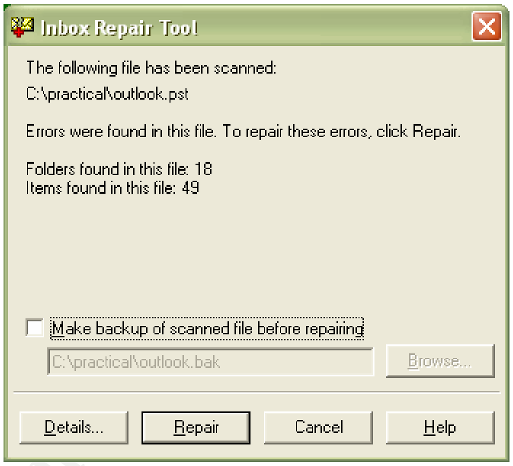


Enter the name of the file you want to scan: C:\practical\outlook.pst

Once repaired, the client program will happily read and use the archive file just as intended.

However, the repair process that scanpst.exe performs assumes there has been significant damage to the file and its internal format. So rather than simply correct the alteration we made above, scanpst.exe assumes that "deleted" items were deleted as part of the corrupting event, so it restores all deleted items it can.

This also illustrates the technique of using client tools and application management tools for forensic purposes rather than esoteric attack techniques in order to recover or revile potential evidence.


Inbox Repair Tool

The following file has been scanned:
C:\practical\outlook.pst

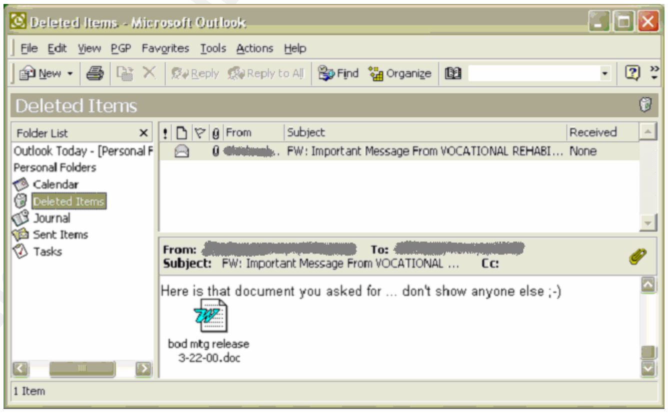
Errors were found in this file. To repair these errors, click Repair.

Folders found in this file: 18
Items found in this file: 49

☐ Make backup of scanned file before repairing
C:\practical\outlook.bak

The repaired .pst is then opened for review, and the previously empty Deleted Items folder now contains a fully available message along with its attachment.

This example may seem contrived, but it is not. People really do *protect* things by saying, "don't show anyone else."


Deleted Items - Microsoft Outlook

[redacted] FW: Important Message From VOCATIONAL REHABI... None

From: [redacted] To: [redacted]
Subject: FW: Important Message From VOCATIONAL ... Cc:

Here is that document you asked for ... don't show anyone else ;-)

bod mtg release 3-22-00.doc

1 Item

Another file of interest is the Dr. Watson dump log.

Below several lines that appear suspicious have been pulled from the Dr. Watson dump log on the possibility they might indicate the presence of malware or attack. Malware or attack is important because, if successful, the inappropriate activity could actually have occurred, though performed by an unauthorized intruder rather than an authorized staff member. The goal of analysis is to discover *what actually happened*, which is a much broader question than simply *did it happen*.

Regardless of any indications of malware, other useful information found in this log includes the dates of the first and last events, 3/17/1999 through 8/14/2001 in this case, along with snapshot views of the processes running on this system, user account names are correlated with date and time stamps (placing a person at this system at specific times).

```
Microsoft (R) Windows NT (TM00 DrWtsn32
Copyright (C) 1985-1996 Microsoft Corp. All rights
reserved.

Application exception occurred:
        App: outlook.dbg (pid=125)) Version 4.
        When: 3/17/1999 @ 16:25:41.126
        Exception number: c0000005 (access violation)

*----> System Information <----*
```

```
        Number of Processors: 1
        Processor Type: x86 Family 5 Model 2 Stepping 12
        Windows Version: 4.0
        Current Build: 1381

*----> Task List <----*
   0 Idle.exe
   2 System.exe
  20 smss.exe
  30 csrss.exe
  34 WINLOGON.exe
  40 SERVICES.exe
  43 LSASS.exe
  69 SPOOLSS.exe
  86 RPCSS.exe
  94 PSTORES.exe
 107 NDDEAGNT.exe
  57 explorer.exe
 113 FINDFAST.exe
 103 MSOFFICE.exe
 105 OSA.exe
 125 OUTLOOK.exe
  64 mapisp32.exe
  90 DRWTSN32.exe
   0  Total.exe
```

The FINDFAST program is reported in operation. This feature of Windows can create a database of files and their contents in order to allow the operator to perform a quick search of the disk by filename or content. Later in the investigation, these database files will be specifically located and analyzed.

Also of interest are OUTLOOK.exe and MSOFFICE.exe because they are common attack vectors of hacker exploits. Malicious scripts (javascript, vbscript, wsh, java) can be embedded in emails and documents and then automatically launched into operation when opened. This is an example of the attack vector these two programs provide and that attackers take advantage of.

```
Application exception occurred:
        App:  (pid=231)
        When: 8/25/1999 @ 10:7:12.112
        Exception number: c0000005 (access violation)

*----> System Information <----*
        Number of Processors: 1
        Processor Type: x86 Family 5 Model 2 Stepping 12
        Windows Version: 4.0
        Current Build: 1381
        Service Pack: 4
        Current Type: Uniprocessor Free

*----> Task List <----*
   0 Idle.exe
   2 System.exe
  20 smss.exe
  30 CSRSS.exe
  34 WINLOGON.exe
  40 SERVICES.exe
  43 LSASS.exe
  71 SPOOLSS.exe
  90 inojobsv.exe
  83 LogWatNT.exe
  96 RPCSS.exe
 104 NDDEAGNT.exe
 117 PSTORES.exe
 119 EXPLORER.exe
 141 systray.exe
 113 realplay.exe
 143 LOADWC.exe
 145 tsystray.exe
 147 realmon.exe
 152 FINDFAST.exe
 154 MSOFFICE.exe
 156 OSA.exe
 170 OUTLOOK.exe
 175 mapisp32.exe
 190 WINWORD.exe
 200 EXCEL.exe
 221 NTVDM.exe
 213 IEXPLORE.exe
 231 spinner.exe
 251 ssstars.scr.exe
 254 DRWTSN32.exe
   0 _Total.exe


*----> Raw Stack Dump <----*
0284f5f8  b4 23 e7 77 ee 00 99 00 - 80 00 00 00 00 00 00 00  .#.w............
0284f608  f2 03 95 02 00 00 00 00 - ad 02 00 00 01 00 00 00  ................
0284f618  79 f7 84 02 80 00 00 00 - 5c 22 42 00 64 f6 84 02  y.......\"B.d...
0284f628  66 6f 42 00 4c f6 84 02 - 6c 27 e7 77 c0 8e 6b 00  foB.L...l'.w..k.
0284f638  80 00 00 00 00 00 00 00 - f2 03 95 02 01 00 00 00  ................
0284f648  66 6f 42 00 00 00 00 00 - c7 43 41 00 ee 00 99 00  foB......CA.....
0284f658  80 00 00 00 00 00 00 00 - f2 03 95 02 95 f8 84 02  ................
0284f668  46 6d 42 00 30 22 42 00 - 75 74 69 6c 2e 63 70 70  FmB.0"B.util.cpp
0284f678  23 36 37 09 2d 41 75 64 - 69 6f 41 64 41 6c 65 72  #67.-AudioAdAler
0284f688  74 0a 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  t...............
0284f698  48 00 00 00 d8 53 f6 77 - 48 05 14 00 3a 07 db 00  H....S.wH...:...
0284f6a8  c4 04 14 00 8c 1c f1 77 - 00 00 14 00 08 20 3e 02  .......w..... >.
0284f6b8  3c 07 db 00 5d 4d e8 01 - 14 00 00 00 01 00 f8 1f  <...]M..........
```

Of interest in this event, an access violation occurred to an unnamed process (with process id 231). Processes running at the time included:

- Inojobsrv.exe
- Realplayer.exe
- Loadwc.exe
- MSWord.exe
- Outlook.exe
- MS Internet Explorer
- Ssstars.exe

Though the cause of many of these types of error events is bad data, each of the .exe files listed here were located using the find command to search through the loopback view of the file system. Once located, their MAC times were checked with the stat tool to verify that they were likely there at the time of this error event; all did. All three MAC times were inspected to make sure they made logical sense and that those time stamps were in sync with other files known to be in the same application distribution.

Each of the programs was then scanned for viruses (again), parsed for visual review using the strings tool. When possible, the md5 hash of each program file was compared to copies retrieved from original CDs. Only a few could be validated this way, but none showed any signs of tampering.

The last entry in the Dr. Watson log was dated 8/14/2001 @ 12:45:47.515, providing more context to the evolving timeline.

Another area to look into is the pagefile.sys, Window’s swap file.

While a registry setting can be set to wipe (over write) the pagefile.sys each time Windows is shut down, that setting is seldom used, which makes the pagefile.sys swap file a potential gold mine of possibly relevant information.

Below is a list of the strings of interest harvested from pagefile.sys.

```
LOGONSERVER=\\SERVER1
COMPUTERNAME=PC69
OS=Windows_NT
rvice Pack 4
ADPA Security Package
DPA
SicSSPCSecret
*-0369<?B
&33nop_$%
@8x8p
sword.
The passwo
```

The strings harvested from pagefile.sys are also a good source of possible passwords. These strings were placed in a text file, cleaned up, and later used as a wordlist for at least two brute force password-cracking tool. In fact, this technique is best performed using the entire strings output of the disk image, again both ASCII and Unicode.

```
C:\practical>sort pagefile.sys.strings > sorted

C:\practical>uniq sorted > uniq

C:\practical>wc *
  14217   22359  364745 pagefile.sys.strings
  14217   22359  364745 sorted
   6474   11585  224679 uniq
  34908   56303  954169 total

C:\practical >cut -b -10 uniq > wordlist
```

Thumbs image database files

Each disk image was searched for files named thumb*, hoping to locate the thumbnail picture database files that Windows stores in directories containing image files. These hidden thumbnail files are then used to support the thumbnail view provided by Windows Explorer and other similar programs. However, no thumbnail files were found.

The Windows SAM file

Even though the entire contents of the disk image is directly available in this analysis environment, it could be helpful to know the logical local administrator password, or other passwords that might provide hints to other passwords. Along with the fact that most user passwords are trivial to crack is the fact that people tend to use the same or similar password across several authentication systems. So it is likely that any passwords that can be discovered will open the way to other password-protected environments.

Note that the guest account has no password. While this is not an intrusion or attack, identifying a well-known attack vector is noteworthy and could become a critical clue, if actual signs of intrusion are discovered.

To do this, the samdump [45] program is fed the raw SAM file and it produces the hash values for each account in that SAM file, in this example, samdump's output is piped to a file which will be fed into John-the-ripper [44]. These hash values are generated with the algorithms in Microsoft's NTLM (NT LAN Manager) authentication system that is native to Windows NT/2k/xp.

```
C:\PRACTICAL>samdump SAM > sam.hash
SAMDump 1.04. Created by Dmitry Andrianov

C:\PRACTICAL>cat sam.hash
Administrator:500:31D6CFE0D16AE931B73C59D7E0C089C0:
00000000000000000000000000000
000:Built-in account for administering the computer/domain::
Guest:501:NO PASSWORD*********************:NO
PASSWORD*********************:Built
-in account for guest access to the computer/domain::
```

With the NTLM (often referred to as LANMAN) hash values acquired, the disk-imaging host (which has been idle since harvesting the original disk image) is given the task of brute force password cracking these hashes using john-the-ripper, a popular and powerful password-auditing tool.

John-the-ripper is given a wordlist file, consisting of the strings to try as a password. In this case, there are 1,402,823 words. This wordlist was created earlier by concatenating the strings output of the disk image to a generic dictionary word list. The resulting file was then sorted and duplicate lines were removed. Before using a list of all strings pulled from the image file, files that would more contain the password in clear-text was used for a much quicker brute-force cracking session. Those files included pagefile.sys, everything from the temp directory, and everything from the recycle bin folders.

After the wordlist has been exhausted, john-the-ripper was launched in its **I**ncremental Mode. As the documentation distributed with john-the-ripper states,

> "This [Incremental mode] is the most powerful cracking mode, it can try all possible character combinations as passwords. However, it is assumed that cracking with this mode will never terminate because of the number of combinations being too large (actually, it

> will terminate if you set a low password length limit, or make it use a smaller character set), and you'll have to interrupt it earlier.”

```
# wc bfwordlist.txt
1402823 1481730 14913986 bfwordlist.txt

# ./john -wordfile:bfwordlist.txt sam.hash

Loaded 2 passwords with no different salts (NT LM DES [24/32 4K])
guesses: 0  time: 0:00:00:03 21%  c/s: 130944  trying: CETERIS - CETUC
guesses: 0  time: 0:00:00:14 100%  c/s: 149899  trying: ZZVOLUM – ZZZZZZZ

# ./john I:Lanman sam.hash
 Loaded 2 passwords with no different salts (NT LM DES [24/32 4K])

guesses: 0   time: 0:01:01:14 c/s 68131 trying: LMINB07 – LCCUMY9
guesses: 0   time: 4:02:43:50 c/s 91814 trying: LPWVBDT – ZLHGLBW
guesses: 0   time: 5:21:48:11 c/s 111045 trying: HJ0PW7U – EJ0PW7U
Session aborted

# ./john status
guesses: 0   time: 5:21:48:12 c/s 111045
```

After 5 days (432000) and 21 hours (75600) and 48 minutes (2880) of password cracking at a rate of 111,045 passwords attempts per second, john-the-ripper had tried over 56 billion possible passwords without finding the correct one.

(432000 + 75600 + 2880) * 111,045 = 56,686,251,600

As the search for evidence continues, in this case with nothing of obvious interest, it is important to understand that evidence does not always consist of a single glaring item. Evidence can also consist of the combination of several seemingly unrelated items, the order or timing of normal items, and the absence of items. Not having found that glaring smoking gun does not indicate anything, until all the relevant data is gathered and put into an objective context.

Verify the integrity of the OS binaries

To verify that the OS binaries (Window NT 4.0) had not been altered or replaced with malicious versions, the md5 hash values of each binaries file (.exe and .dll) in the WINNT\SYSTEM32 folder was compared to the known good hash values from a system freshly installed from the original CDs.

The first round showed that none of the hash values matched. After installing the first service pack for NT 4.0, many did. Installing the first service pack to MSIE 5.x created a few more matches. It had not been expected that all files could be validated this way due to the large number of patches issued by Microsoft and the haphazard way/order they could have been applied - the permutations made finding a complete match improbable. Still enough of the core operating system was validated that a reasonable level of trust could be established that the integrity of the operating system being analyzed had not been compromised (other than by sloppy programming).

Guarded trust was reasonable because NT/Win2k Trojan programs have only recently appeared in public, and this system was in production until 2001, and not after.

Trojan files are “hacked” versions that replace the authentic copies and then provide all of the expected services, while masking all clues to the presence of an intruder and possible providing additional services to an intruder. For example, a trojaned login.exe program will function properly just as the user expects. But in addition, it may also send the typed user names and passwords on to the intruder.

Scanning for viruses, spyware, adware, and backdoors

While the people who actually used this hard disk may not have done anything inappropriate, it is possible that the system had been successfully compromised without their knowledge. A virus/worm could have infected the system, an intruder could have created a backdoor or installed trojan software, or some type of spyware/adware could be gathering information for any number of purposes.

Each of these possibilities is intended to function without making their activity known. So, there are two types of inappropriate activity we are looking for – one performed by a person who is authorized, and the other, by a person who is not. This means there are two distinct types of activity, each with different methods of obscuring, hiding, or destroying the indications of that activity.

So, at this point, the system is “scanned” for signs of external intrusion using the same types of methods employed in Incident Handling.

It would damage the investigation’s integrity to “activate” the operating system contained in the disk image and have it scan its own files. Instead, the files are kept inert and are scanned remotely from a trusted operating system. A convenient side effect is that, since the files being scanned are on a read-only file system, a virus scanner cannot alter any tainted files it might identify.

More specifically, while operating in the Linux operating system, the disk image file is mounted and its contents are copied from the mount point into a regular folder on the Linux system (ext2 file system in this case). The computer is then booted into Windows, where the Linux ext2 partition is mounted in read-only mode. At this point, the Windows operating systems and the applications running under it have read-only access to the logical files and their contents. This allows the Windows based malware scanners to interrogate the files from the disk image (again without being able to alter them).

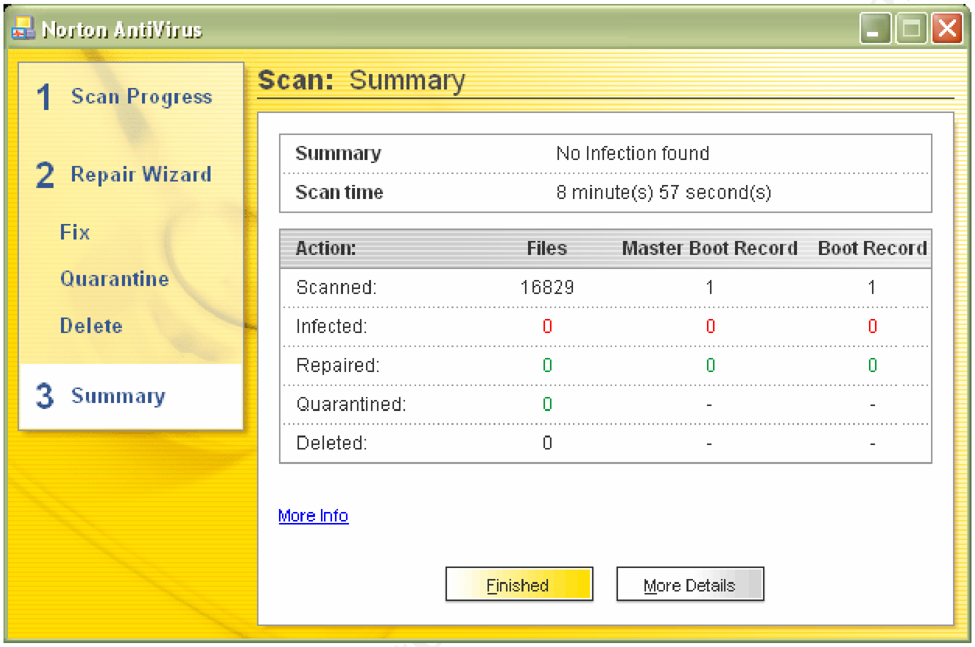
An odd problem occurred while scanning for computer viruses. Each virus scanner used reported read errors soon after the scan began. The virus scanners and their virus signature files were all current and functioned properly when scanning other disks (removable and not). Both virus scanners, Symantec (Symantec 59) and McAfee (McAfee 60) AVS

(Anti-Virus Scanners) failed, but at different points during their scans.

After some troubleshooting, what finally worked was while in Linux, burn the Linux folder containing these files onto a new blank CD, then switch to Windows and scan the CD for viruses and other malware.

Rather than launch the scans from the GUI interfaces, the command line was used to pass several parameters directly to the scanners. Below is an example of running Norton's AntiVirus program in this way.

```
C:\PRACTICAL>navw32 D:\disk1.fs\ /B- /S+ /HEUR:3
```



Scan: Summary

Summary	No Infection found
Scan time	8 minute(s) 57 second(s)

Action:	Files	Master Boot Record	Boot Record
Scanned:	16829	1	1
Infected:	0	0	0
Repaired:	0	0	0
Quarantined:	0	-	-
Deleted:	0	-	-

Even though the files harvested from the disk image did not seem to have any virus infection, the C:\Alert folder on the disk image contained 102 "event data" files. These appear to have been generated by the system's antivirus, one file per event.

These event files were concatenated into a single file for review using the strings, grep, and tail tools. This showed that there had been 50 separate incidents of virus infection identified by the local antivirus software. The original dates for these files ranged from Oct 6, 1999 to Mar 23, 2000.

```
C:\PRACTICAL>strings -a dat.files | grep -c "detected"
50

C:\PRACTICAL>strings -a dat.files | grep -c "Melissa"
50

C:\PRACTICAL>strings -a dat.files | grep "Melissa" | tail –n 1
Real-time on MyHost: The W97M/Melissa virus was detected in

C:\INOCULAN\VIRUS\SISTER PRAYER.DOC.
Machine: MyHost.
```

```
User: ServerNT\joeuser,
Action: Canceled - Virus is in the Move Directory.
```

The process illustrated above was to search the entire group of log files for what appeared in the first instance to be a keyword that identified actual infection events and excluded normal housekeeping and maintenance reports.

Seeing that the first event had to do with the Melissa virus, Melissa was used as the keyword in the next search. In both, there were 50 lines that matched. By displaying the last line of the search results, the details of the event are shown. This is also an example of chaining tools together to produce granular results. The strings tool parses out the ASCII encoded (-a) strings from the dat.files file, passing the results to grep to filter out the lines that did not include the case sensitive string “Melissa,” and finally, those results are passed to the tail tool, which then displays on-screen only the last line (-n 1) of the results.

By listing the files in this folder with the ls tool, sorted in reverse order (-r) by date (-T) in a moderately detailed format (-o) the date range is clearly shown. To verify this, the stat tool is run on these same files to verify the MAC time stamps that ls is reporting.

```
C:>ls *.dat –tor
-r--r--r--   1 user          328 Oct  6  1999 ev2.dat
-r--r--r--   1 user          328 Oct  6  1999 ev1.dat
(Most lines omitted for readability)
-r--r--r--   1 user          328 Mar 23  2000 ev108.dat
-r--r--r--   1 user          328 Mar 23  2000 ev107.dat
```

2. Examine file system for back doors, check for setuid and setgid files

The first task was to identify the user accounts. However, the account metadata and file permissions in a Windows system (this is actually a file system issue, rather than an operating system issue, but that line, normally very clear in Linux, is very blurred in the Windows world) are organized and managed very differently than in Linux.

In effect, the user account metadata is washed out when the file system is viewed through Linux. That metadata is still present in the disk image, but is not properly represented by Linux tool. This is not a failure of the Linux environment; instead it reflects an inability to translate one logical data structure to another. This is the basic “fitting a square peg into a round hole” dilemma. Both make sense in its own context, but the elements of one do not necessarily align with the elements of the other.

From the information gathered below, correlating the MAC times of the account folders with other files in the system implies the user account owns a given file. This approach is fine when constructing an approximation, but comparing them to the metadata in the actual file system would validate these findings.

This can be accomplished by writing the disk image file (the byte-for-byte representation of the suspect disk) to a clean hard disk that is then used to boot a test system. Booting with this disk will alter the time stamps and log file contents all over the file system, not to mention the registry, but it would create an environment where an analyst could query the file system for owner, group, permission, and restriction information.

```
# ls -to
total 0
dr-x------    1 root         4096 Mar 20  2000 joeuser
dr-x------    1 root         4096 Jun 25  1999 Default User
dr-x------    1 root         4096 Feb 25  1999 Administrator
dr-x------    1 root            0 Feb 25  1999 All Users

# ls RECYCLER/ -tro --time=ctime
total 0
dr-x------    1 root        45056 Mar  1  1999 S-1-5-21-72424920-1370634877-1703228666-1254
dr-x------    1 root         4096 Mar  1  1999 S-1-5-21-72424920-1370634877-1703228666-500
dr-x------    1 root         4096 Mar 20  2000 S-1-5-21-72424920-1370634877-1703228666-1148
dr-x------    1 root         4096 Sep 29  2000 S-1-5-21-72424920-1370634877-1703228666-1385

# stat *
  File: "S-1-5-21-72424920-1370634877-1703228666-1148"
  Size: 4096            Blocks: 0          IO Block: 4096   Directory
Device: 700h/1792d      Inode: 7209        Links: 1
Access: (0500/dr-x------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Tue Nov  6 16:29:31 2001
Modify: Tue Nov  6 16:29:03 2001
Change: Mon Mar 20 16:19:26 2000

  File: "S-1-5-21-72424920-1370634877-1703228666-1254"
  Size: 45056           Blocks: 0          IO Block: 4096   Directory
Device: 700h/1792d      Inode: 3825        Links: 1
Access: (0500/dr-x------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Sat Sep 22 12:24:03 2001
Modify: Wed Mar 15 15:16:52 2000
Change: Mon Mar  1 11:22:34 1999

  File: "S-1-5-21-72424920-1370634877-1703228666-1385"
  Size: 4096            Blocks: 0          IO Block: 4096   Directory
Device: 700h/1792d      Inode: 16358       Links: 1
Access: (0500/dr-x------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Sat Sep 22 12:24:03 2001
Modify: Fri Sep 29 13:45:10 2000
Change: Fri Sep 29 13:40:38 2000

  File: "S-1-5-21-72424920-1370634877-1703228666-500"
  Size: 4096            Blocks: 0          IO Block: 4096   Directory
Device: 700h/1792d      Inode: 3828        Links: 1
Access: (0500/dr-x------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Sat Sep 22 12:24:03 2001
Modify: Wed Jan 24 16:22:41 2001
Change: Mon Mar  1 11:38:03 1999
```

```
C:\practical>regdmp -h software HKEY_USERS > software.users

C:\practical>grep -i -f sids.txt -A 5 -B 5 software.users
```

```
(Only a few relevant lines are displayed below.)

ProfileImagePath = REG_EXPAND_SZ %SystemRoot%\Profiles\Default User
SCentralProfile = S-1-5-21-72424920-1370634877-1703228666-1148

ProfileImagePath = REG_EXPAND_SZ %SystemRoot%\Profiles\All Users
CentralProfile = S-1-5-21-72424920-1370634877-1703228666-1254
    S
ProfileImagePath = REG_EXPAND_SZ %SystemRoot%\Profiles\joeuser
CentralProfile = S-1-5-21-72424920-1370634877-1703228666-1385

ProfileImagePath = REG_EXPAND_SZ %SystemRoot%\Profiles\administrator
CentralProfile = S-1-5-21-72424920-1370634877-1703228666-500
```

From this information, we can assume that:

```
Administrator     = S-1-5-21-72424920-1370634877-1703228666-500
All Users         = S-1-5-21-72424920-1370634877-1703228666-1254
joeuser           = S-1-5-21-72424920-1370634877-1703228666-1385
Default User      = S-1-5-21-72424920-1370634877-1703228666-1148
```

These associations hinge on the fact that a Windows administrator is always 500, while the user account numbers increment beginning at 1000. By associating the account number with a name, and the date of each account's RECYCLER folder (container for deleted files) to the date of the folders in the \Profiles folders (there is only one profile per user account), and noting the incrementing pattern of account numbers you begin to get a picture of whose deleted files are in while RECYCLER folder. Even so, these assumptions could be incorrect, especially if the system is actually used by several different accounts.

3.Examine file system for any sign of a sniffer program

To determine if a network sniffer was present on the system, specific strings were looked for on the entire disk, not just the regular files.

In other steps, something unambiguous is searched for, such as a keyword, items in log and history files, or files of a specific content (pornographic pictures). In this case, no one characteristic will clearly identify a sniffer program. Here we want to look for a program file that has a specific purpose, in this case covertly capturing data that an intruder can use to their advantage. But that type of program/condition does not necessarily fit into any of the categories (keywords or size) that can easily be searched for. If filenames alone are searched, then a sniffer program that is renamed will not be identified. Unless there is a clear-text string such as "PROMISCUOUS MODE" or "sniffer", string searching will not help.

It may be possible to develop a signature database that identifies binary material unique to sniffer programs, such as the binary representation of the code used to set a NIC into promiscuous mode. However, there are legitimate reasons for putting a NIC into promiscuous mode, and not all programs used to do this are in binary format. Pretty soon, the return on investment (ROI) of time and effort outweighs the possible benefit.

Instead a three-pronged approach would seem to be more effective. In this approach are three vectors to walk through when looking for a type of program.

1. what is it; its name
2. what it contains; strings/keywords
3. what it does: executes on the local system

The first step was to search the file inventory for the relevant filenames or fragments. The same technique was used here as in many other steps in this paper: a text file was created containing relevant keywords, and the grep tool was used that list of strings to search the file inventory and report any matches. This produced no findings, which was not surprising. Rename a file from its easy-to-identify name to a name that is unlikely to attract attention is a common stealth technique.

The second tactic is to search the entire disk image for keywords that indicate the existence of a sniffer. Since we are looking for a sniffer, the keyword "promiscuous" is probably the most identifiable. Network sniffers often rely on putting the network adapter into promiscuous mode in order to observe and capture all network traffic. Finding a network card to be in promiscuous mode is unusual, so any such reference would an important clue.

Here, each file in the file system is checked for the "promiscuous" keyword. However, each hit appears to be benign. This same process was then performed on the disk image so that its unallocated sectors (deleted files) were also checked.

While this is redundant (searching the disk image includes searching the files already checked), splitting this task this way will quickly identify an existing file of interest. If there were no findings in this first scan, but the keyword "promiscuous" was found in the second grep, it would indicate that the suspicious file had been deleted.

```
C:\PRACTICAL>grep -i --recursive "promiscuous" d:\*

d:\WINNT/$NtServicePackUninstall$/oemnadma.inf:       else-ifstr(i) $(ValueName) == "PromiscuousModeX"

d:\WINNT/$NtServicePackUninstall$/oemnadma.inf:                   {Promiscuou sModeX, $(NoTitle),
$(!REG_VT_DWORD), $(StatsFlag)})

d:\WINNT/$NtServicePackUninstall$/oemnadma.inf:        DeleteRegValue $(KeyParameters) PromiscuousModeX

d:\WINNT/system32/OEMNADMA.INF: else-ifstr(i) $(ValueName) == "PromiscuousModeX"

d:\WINNT/system32/OEMNADMA.INF:                  {PromiscuousModeX, $(NoTitle), $(!REG_VT_DWORD),
$(StatsFlag)})

d:\WINNT/system32/OEMNADMA.INF: DeleteRegValue $(KeyParameters) PromiscuousModeX

Binary file d:\WINNT/system32/glossary.hlp matches
```

The third tactical prong is to investigate the points where an executable would have the opportunity to automatically go into execution. It is likely that a program would be automatically launched from one of a few specific points. These include; the startup folder, a registry key (such as RUNONCE), the autoexec.bat/nt file, or by a program that is itself automatically launched by a program at one of these points.

So the batch files automatically launched at boot-up and the registry keys that run programs will be checked. Any programs that are automatically launched will be validated.

At the end of this investigation, no indications of a sniffer program had been found.

4. Internet history file and other history files

Initially, the standard history files were examined, but, later in the analysis when reviewing the strings harvested from the disk image, it was discovered that a local proxy program had been installed and had logged a large amount of activity relevant to this study. But for now, the stock history files will be addressed.

Each time a user visits a web site, Windows operating system records that event in a file named index.dat. Actually, there are usually several index.dat files scattered throughout the disk, making it necessary to search the entire file system to identify every instance.

There is always the possibility that a person might rename the file, but that is unlikely. Most people are not familiar enough with their Windows system to know the index.dat files exist or how they are used. Those who are that familiar would simply switch to a browser that does not covertly record and hide the person's activities, but rather allows the person to manage that information.

One trivial use for these records is to identify for the browser which hyperlinks, the blue underscored text, should instead be colored as purple underscored text as a visual indicator that the link has recently been visited.

To perform this task, the file inventory was searched for every instance of a file named index.dat. Once identified, the full path of each file was recorded in a batch file.

```
C:\practical>grep -i "index\.dat" disk1.file.inventory
-r--------    1 root       32768 2001-10-13 09:53:31 -0400 index.dat
-r--------    1 root       49152 2001-10-13 09:53:32 -0400 index.dat
-r--------    1 root       32768 2001-01-11 15:14:38 -0500 index.dat
-r--------    1 root       32768 2001-01-11 15:14:38 -0500 index.dat
-r--------    1 root      147456 2001-10-13 09:53:31 -0400 index.dat
-r--------    1 root      147456 2001-10-13 09:53:31 -0400 index.dat
-r--------    1 root      114688 2001-12-03 14:29:49 -0500 index.dat
-r--------    1 root     2654208 2001-12-03 14:31:26 -0500 index.dat
-r--------    1 root       49152 2001-11-27 08:07:59 -0500 index.dat
-r--------    1 root       65536 2001-11-26 11:31:11 -0500 index.dat
-r--------    1 root       49152 2001-11-27 08:07:59 -0500 index.dat
-r--------    1 root       65536 2001-11-26 11:31:11 -0500 index.dat
-r--------    1 root       65536 2001-12-03 09:48:45 -0500 index.dat
-r--------    1 root       65536 2001-12-03 14:31:32 -0500 index.dat
-r--------    1 root       65536 2001-12-03 14:31:32 -0500 index.dat
-r--------    1 root       65536 2001-12-03 09:48:45 -0500 index.dat
-r--------    1 root     1736704 2001-12-03 08:45:00 -0500 index.dat
-r--------    1 root     1736704 2001-12-03 08:45:00 -0500 index.dat
```

There each line is expanded into a command to run the iehist.exe program on each of these files and redirect its output to a single text file named url.history.recover for further review.

```
iehist "l:\root\morgue\disk1\WINNT\Profiles\joeuser2\Cookies\index.dat" >> url.history.recover
iehist "l:\root\morgue\disk1\WINNT\Profiles\joeuser2\History\index.dat" >> url.history.recover
iehist

(lines omitted)

iehist "l:\root\morgue\disk2\Windows\History\History.IE5\MSHist012001090720010908\index.dat" >>
url.history.recover
```

This batch file was then launched. As it ran, iehist.exe reported the number of URLs identified in each index.dat file. By running the wc (word count) program on the url.history.recover report file, it was determined that there were 8,108 URLs identified in this process.

```
C:\practical>recover.url.history.cmd

C:\practical>iehist "l:\root\morgue\disk1\WINNT\Profiles\joeuser2\Cookies\index.dat
Urls retrieved 6

C:\practical>iehist "l:\root\morgue\disk1\WINNT\Profiles\joeuser2\History\index.dat"
1>>url.history.recover
Urls retrieved 7

(lines omitted)

C:\practical>iehist
"l:\root\morgue\disk2\Windows\History\MSHist012000070620000707\index.dat"
1>>url.history.recover
Urls retrieved 3

C:\practical>iehist
"l:\root\morgue\disk2\Windows\History\History.IE5\MSHist012001090720010908\index.dat"
1>>url.history.recover
Urls retrieved 75

C:\practical>wc url.history.recover
   8108   17886  833146 url.history.recover
```

Of the 8,108 URLs recorded, 1,262 of them did not contain a date time stamp. These were redirection records, which occur when a web page automatically pointing the browser to another page that is to be displayed in its place. The remaining 6847 contained date time stamps.

Since each record was dated, sorting the list by the date field revealed the range of dates for these records. The date range was 1998/12/16 through 2001/12/3.

Having gathered a detailed listing of visited URLs, a search could be performed to identify any keywords that would indicate directions for further investigation. Several lists of keywords were individually fed to grep, each focusing on a type of inappropriate activity, including; online gambling, predatory/stalking/threatening, in-house proprietary data/resources, and others. None identified anything of interest, except the list of sexual related terms. While this step identified only a relatively few number of "hits," it established a type of inappropriate activity, which could be investigated further. In fact, in later steps, significant findings were discovered.

```
# cat grepterms
sex
slut
singles
adult

# grep -i -f grepterms url.history.recover
URL|2001/10/31
17:47:45|http://dps1.travelocity.com/airgcobrand.ctl?smls=Y&Service=YHOE&.resform=YahooF
lightsR&.intl=us&trip_option=roundtrp&module=calendar&adult_pax_cnt=1&chld_pax_cnt=0&sen
ior_pax_cnt=0&dep_arp_cd%281%29=Louisville&arr_arp_cd%281%29=Nevada&dep_dt_mn_1=Nov&dep_
dt_dy_1=21&dep_tm_1=9%3a00am&dep_dt_mn_2=Nov&dep_dt_dy_2=23&dep_tm_2=5%3a00pm&cls_svc=YR
&pref_aln=all&aln_cd%281%29=&aln_cd%282%29=&aln_cd%283%29=&num_cnx=2&tm_range=+&SEQ=&dep
_arp_range%281%29=150&arr_arp_range%281%29=150
URL|2001/11/27 20:52:22|http://us.rmi.yahoo.com/js/postproc/singleSignon.64585.js
URL|2001/11/27 19:50:13|https://us.rmi.yahoo.com/js/postproc/singleSignon.64585.js
REDR|
|http://www.hit.stats4all.com/asp/hit.asp?sSiteName=superlaugh&nav=MSIE&version=400&scre
ensize=800*600&colors=16&sver=13&java=y&rf=bookmark&navlan=&plug=&sUrl=file%3A//C%3A%5CW
INDOWS%5CTEMP%5CSuperLaugh%2520Greetings%2520and%2520Fun%2520Pages.htm&sExtra=None
```

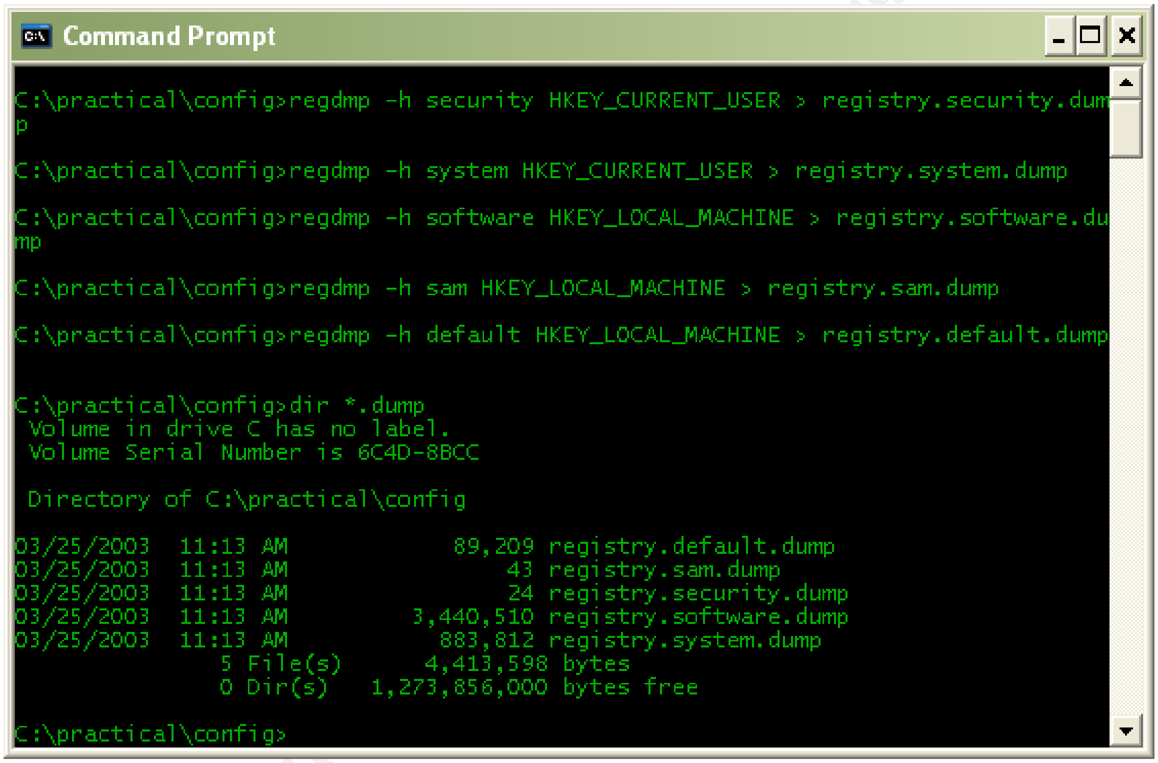
This was noteworthy, but not quite a smoking gun or body on the floor.

5.System Registry or /proc examination

Once the disk image of the NT boot volume has been mounted, the complete contents of the winnt\system32\config folder are available for inspection. The files are the system's actually registry files. While normally protected by the operating system, there is no such protection when the operating system is not running and in full control of its resources, such as the registry.

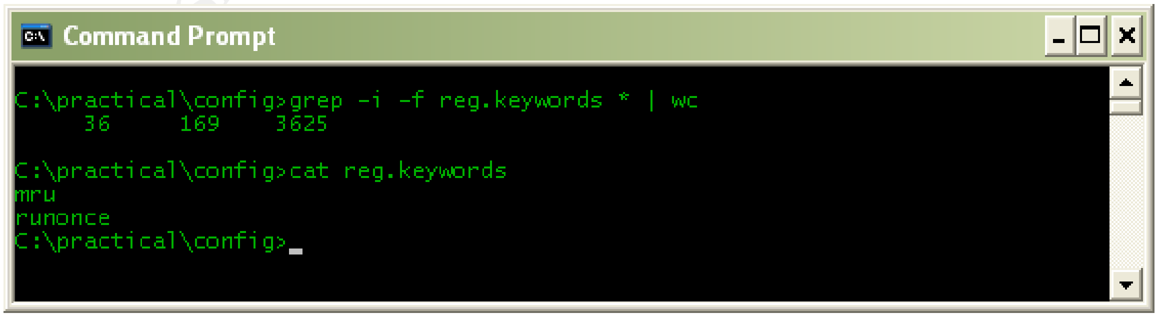
```
C:\practical>dir
 Volume in drive C has no label.
 Volume Serial Number is 6C4D-8BCC

 Directory of C:\practical

04/03/2003  03:16 PM    <DIR>          .
04/03/2003  03:16 PM    <DIR>          ..
04/01/2003  03:03 PM           393,216 AppEvent.Evt
04/01/2003  03:03 PM            86,016 default
04/01/2003  03:03 PM             1,024 default.LOG
04/01/2003  03:03 PM            81,920 default.sav
04/01/2003  03:03 PM                 0 default.tmp.LOG
04/01/2003  03:03 PM                 0 DEFAUL~1.LOG
04/01/2003  03:03 PM                 0 deftnew.LOG
04/01/2003  03:03 PM            16,384 SAM
04/01/2003  03:03 PM             1,024 SAM.LOG
04/01/2003  03:03 PM            16,384 sam.sav
04/01/2003  03:03 PM            65,536 SecEvent.Evt
04/01/2003  03:03 PM            32,768 SECURITY
04/01/2003  03:03 PM             1,024 SECURITY.LOG
04/01/2003  03:03 PM            28,672 security.sav
04/01/2003  03:03 PM                 0 softnew.LOG
04/01/2003  03:03 PM         6,193,152 software
04/01/2003  03:03 PM             1,024 software.LOG
04/01/2003  03:03 PM         4,382,720 software.sav
04/01/2003  03:03 PM                 0 software.tmp.LOG
04/01/2003  03:03 PM                 0 SOFTWA~1.LOG
04/01/2003  03:03 PM           393,216 SysEvent.Evt
```

```
04/01/2003  03:03 PM         1,306,624 system
04/01/2003  03:03 PM         1,306,624 SYSTEM.ALT
04/01/2003  03:03 PM             1,024 system.LOG
04/01/2003  03:03 PM         1,228,800 system.sav
04/01/2003  03:03 PM                 0 system.tmp.LOG
04/01/2003  03:03 PM                 0 SYSTEM~1.LOG
04/01/2003  03:03 PM                 0 systnew.LOG
04/01/2003  03:03 PM            77,824 userdiff
04/01/2003  03:03 PM            90,112 userdifr
04/01/2003  03:03 PM             1,024 userdifr.LOG
              31 File(s)     15,706,112 bytes
               2 Dir(s)   2,164,035,584 bytes free
```



```
C:\practical\config>regdmp -h security HKEY_CURRENT_USER > registry.security.dum
p

C:\practical\config>regdmp -h system HKEY_CURRENT_USER > registry.system.dump

C:\practical\config>regdmp -h software HKEY_LOCAL_MACHINE > registry.software.du
mp

C:\practical\config>regdmp -h sam HKEY_LOCAL_MACHINE > registry.sam.dump

C:\practical\config>regdmp -h default HKEY_LOCAL_MACHINE > registry.default.dump


C:\practical\config>dir *.dump
 Volume in drive C has no label.
 Volume Serial Number is 6C4D-8BCC

 Directory of C:\practical\config

03/25/2003  11:13 AM            89,209 registry.default.dump
03/25/2003  11:13 AM                43 registry.sam.dump
03/25/2003  11:13 AM                24 registry.security.dump
03/25/2003  11:13 AM         3,440,510 registry.software.dump
03/25/2003  11:13 AM           883,812 registry.system.dump
               5 File(s)      4,413,598 bytes
               0 Dir(s)   1,273,856,000 bytes free

C:\practical\config>
```

The grep tool was then used to check these dump files for registry entries of interest. Of the 36 reported, none appeared to be of any interest.



```
C:\practical\config>grep -i -f reg.keywords * | wc
     36     169    3625

C:\practical\config>cat reg.keywords
mru
runonce
C:\practical\config>_
```

6. Show start up files and processes

The illustrations below are of the files and links that are activated each time this Windows system is booted. The boot process is well understood, so dissecting it is straightforward. In addition to the system wide items, each account can activate programs and settings unique to it.

The autoexec.bat script contains three lines having to do with the sound blaster audio card. Note that the original host contained a sound blaster audio card, so these lines would appear benign.

C:\AUTOEXEC.BAT

```
SET SOUND=C:\PROGRA~1\Creative\CTSND
SET MIDI=SYNTH:1 MAP:E
SET BLASTER=A220 I5 D1 H5 P330 T6
```

The config.sys file existed, but had no content.

C:\CONFIG.SYS

```
Empty
```

The joeuser account was the only one of interest. In its startup folder, several items were automatically launched when this user logged onto the system.

```
[/WINNT/Profiles/All Users/Start Menu/Programs/Startup] # ls -o
total 0
-r--------    1 root         444 Aug 13  1999 InoculateIT Realtime Monitor.lnk
-r--------    1 root         532 Mar  1  1999 Microsoft Find Fast.lnk
-r--------    1 root         514 Feb 25  1999 Office Startup.lnk

[/WINNT/Profiles/joeuser/Recent]# ls -o
total 0
-r--------    1 root         473 Nov  6  2001 2nd request.lnk
-r--------    1 root         473 Nov  2  2001 # 2 request.lnk
-r--------    1 root         471 Aug 14  2001 addresses.lnk
-r--------    1 root         445 Nov  6  2001 address.lnk
-r--------    1 root         471 Aug 15  2001 August 11.lnk
-r--------    1 root         471 Aug 15  2001 August 13.lnk
-r--------    1 root         435 Aug 15  2001 db1.lnk
-r--------    1 root         452 Oct 19  2001 employee notification main office.lnk
-r--------    1 root         452 Nov  6  2001 May 25.lnk
-r--------    1 root         410 Oct  9  2001 New Microsoft Excel Worksheet.lnk
-r--------    1 root         501 Sep 20  2001 nofind form letter.lnk
-r--------    1 root         481 Nov  6  2001 nofind letter.lnk
-r--------    1 root         317 Sep 20  2001 Rescued Document.lnk
-r--------    1 root         467 Aug 14  2001 Vows etc.lnk
-r--------    1 root         503 Aug 15  2001 wedding addresses.lnk

[/WINNT/Profiles/joeuser/Start Menu/Programs/Gnucleus]# ls -o
total 0S
-r--------    1 root         455 Jul 27  2001 Downloads.lnk
-r--------    1 root         459 Jul 26  2001 Gnucleus.lnk
-r--------    1 root         463 Jul 26  2001 Uninstall.lnk

[/joeuser/Start Menu/Programs/Gnucleus]# cat Gnucleus.lnk
L?F? `|????`????`|???????O? ?:i?+00?#C:\?%1?*:~Program
```

```
FilesPROGRA~11?*<~Gnucleus2??*f& Gnucleus.exeU-T?!g?C:\Program
Files\Gnucleus\Gnucleus.exe5..\..\..\..\..\..\Program Files\Gnucleus\Gnucleus.exeC:\Program
Files\Gnucleus

[/joeuser/Start Menu/Programs/Gnucleus]# cat Downloads.lnk
L?F??D?????? }??`?A????O? ?:i?+00?#C:\?%1?*:~Program
FilesPROGRA~11?*<~Gnucleus!1?*<~DownloadsDOWNLO~1R-Q?!g?C:\Program
Files\Gnucleus\Downloads2..\..\..\..\..\..\Program Files\Gnucleus\DownloadsC:\Program
Files\Gnucleus
```

One item that jumps out is the Gnuclus program. Gnuclus is an open-source "Gnutella" Client.

> What is Gnutella?
> Posted by adminAngelo Sotira on December 3, 2001 at 7:47 PM
> *Since its inception, the entire premise of the Internet centered on file sharing. ... To put it simply, Gnutella puts the personal interaction back into the Internet. When you run Gnutella software and connect to the Gnutella Network, you bring with you the information you wanted to make public. And you choose what information to share. You can choose to share nothing; you can choose to share one file, a directory, or your entire hard drive (we do not recommend this option).* (http://www.gnutella.com/)

Oddly, this was posted about 5 hours after this system was shutdown for the last time, 2001-12-03 14:47.

This shows that the system was configured to share its disk space across the Internet with other Gnutella clients. However, when looking back at the list of processes in the Dr. Watson system dump files, there is no indication that Gnuclus was active.

In addition, the dates of the .LNK files (July 2001) are near the end of this system's operational life. That implies that Gnuclus was not operational for most of the system's life, and that any vulnerabilities Gnuclus may have created did not exist during the life of most of the material being analyzed.

Timeline Analysis

The goal of a timeline is to illustrate a person's activities; more accurately, the use the system was put to over a period of time, to prove or disprove a pattern or event.

This timeline was initially created with the fls tool. The bulk of that initial report represented normal operations, which have been omitted. Additional entries were gathered from different sources throughout this analysis.

Timeline

```
The first dated item representing installation of the OS.
Sun Oct 13 1997 21:38:00        3 m.. -/-rwxrwxrwx 48       0        1028-128-1
/WINNT/WINFILE.INI

34 lines of files in the same folder omitted.

Mon Oct 14 1997 01:30:50    43539 m.. -/-rwxrwxrwx 48       0        409-128-4
```

```
/WINNT/system32/oemnade2.inf (deleted-realloc)
```

1300 lines representing the end of the OS installation process.

Below is the first indication of the user accounts.

```
Mon Oct 14 1997 02:38:00      829 m.. -/-rwxrwxrwx 48       0        1835-128-5
/WINNT/Profiles/joeuser/Start Menu/Programs/Accessories/Hyperterminal/MCI Mail.ht
                              945 m.. -/-rwxrwxrwx 48       0        4306-128-5
/RECYCLER/S-1-5-21-72424920-1370634877-1703228666-1254/DC844.HTM (deleted-realloc)
                              829 m.. -/-rwxrwxrwx 48       0        1470-128-4
/WINNT/Profiles/Default User/Start Menu/Programs/Accessories/Hyperterminal/AT&T Mail.ht
                              829 m.. -/-rwxrwxrwx 48       0        1769-128-5
/WINNT/Profiles/Administrator/Start
Menu/Programs/Accessories/Hyperterminal/CompuServe.ht
                              829 m.. -/-rwxrwxrwx 48       0        4328-128-5
/RECYCLER/S-1-5-21-72424920-1370634877-1703228666-1254/DC847.HTM (deleted-realloc)
                              945 m.. -/-rwxrwxrwx 48       0        1474-128-4
```

A full year of uneventful lines omitted.

The installation of NT service pack 4

```
Thu Oct 15 1998 12:04:00   166672 m.. -/-rwxrwxrwx 48       0        9138-128-4
/WINNT/$NtServicePackUninstall$/winsrv.dll
```

(Several lines omitted)

This represents the completion of installing NT SP4

```
Thu Oct 15 1998 17:15:12   102316 m.. -/-rwxrwxrwx 48       0        9315-128-4
/WINNT/$NtServicePackUninstall$/readme.sp4
```

On the following date, there was apparently a significant file system / disk event. The highly protected system files used to manage the file system were recreated.

```
Thu Feb 25 1999 02:56:46        0 mac -/-r-xr-xr-x 48       0        8-128-2   /$BadClus
                                0 mac -/-r-xr-xr-x 48       0        9-128-1   /$Quota
                       1055899648 mac -/-r-xr-xr-x 48       0        8-128-1
/Program Files/Microsoft Office/Office/Forms/CLAP.WMF:$Bad (deleted-realloc)
                          4194304 mac -/-r-xr-xr-x 48       0        2-128-1   /$LogFile
                             8192 mac -/-r-xr-xr-x 48       0        7-128-1   /$Boot
                                0 mac -/-r-xr-xr-x 48       0        3-128-3   /$Volume
                       1055899648 mac -/-r-xr-xr-x 48       0        8-128-1
/$BadClus:$Bad
                           257792 mac -/-r-xr-xr-x 48       0        6-128-1   /$Bitmap
                            36000 mac -/-r-xr-xr-x 48       0        4-128-1   /$AttrDef
                           131072 mac -/-r-xr-xr-x 48       0        10-128-4  /$UpCase
                             4096 mac -/-r-xr-xr-x 48       0        1-128-1   /$MFTMirr
                         20984832 mac -/-r-xr-xr-x 32       0        0-128-7   /$MFT
                                0 mac -/-r-xr-xr-x 48       0        8-128-2   /Program
Files/Microsoft Office/Office/Forms/CLAP.WMF (deleted-realloc)
```

40 minutes later, the system is "worked on" via a telnet session.

```
Thu Feb 25 1999 03:35:19     1253 m.. -/-rwxrwxrwx 48       0        1754-128-4
/WINNT/Profiles/Administrator/Start Menu/Programs/Accessories/Telnet.lnk
```

Worked continued until 2 pm later that same day.

```
Thu Feb 25 1999 15:57:42 42448896 m.. -/-rwxrwxrwx 32       0        10347-128-0
/Program Files/State Business Directory/B/ba_ph.99
```

This is the first dated event in the Dr. Watson error log.

```
Application exception occurred:
        App: outlook.dbg (pid=125)) Version 4.
        When: 3/17/1999 @ 16:25:41.126
        Exception number: c0000005 (access violation)
```

Six months of regular activity lines omitted.

Another NT service pack installed

```
Thu Aug 05 1999 09:14:55       56 m.c d/dr-xr-xr-x 48       0        8973-144-9
/WINNT/$NtServicePackUninstall$
```

Again from Dr. Watson, an error occurred.

```
Application exception occurred:
        App:  (pid=231)
        When: 8/25/1999 @ 10:7:12.112
        Exception number: c0000005 (access violation)
```

The first dated entry in the INOCULAN C:\ALERTS log folder.

```
Oct  6  1999 ev2.dat
```

In the next section of this paper, evidence will be found of browsing pornographic web sites. The entries in the WinProxy cache gave dates when pornographic websites were visited. The following example illustrates the weekly pattern of habitual activity. While there was some diviation, the pattern was to usually spend about 45 minutes, beginning at 4 P.M. most weekdays, and spending the most time on Fridays. This pattern held for just over 18 months. Using 260 minutes per week as an average, over a 72-month period this person spent approximately 18,720 minutes, or **312 hours** (39 work days) browsing pornographic websites.

```
First date; 2000-01-10
       Monday 4:00-4:41       41 min  (2000-01-10 16:01:56 until 2000-01-10 16:41:58)
       Tuesday 10:20-11:02    42 min  (2000-01-11 10:20:04 until 2000-01-11 11:02:32)
       Thursday 4:25-4:37     12 min  (2000-01-13 16:24:00 until 2000-01-13 16:37:40)
       Friday 1:45-4:30       165 min (2000-01-14 15:45:26 until 2000-01-14 16:31:56)
Last date; 2001-07-10
```

The last dated entry in the INOCULAN C:\ALERTS log folder. It would appear that soon after this, the system was switched over to another virus protection product.

```
Jul 23  2001 ev107.dat
```

Here Gnucleus, a Guntilla client is installed. From this point the file system is available to unknown Internet based clients.

```
Jul 26  2001 Gnucleus.lnk
```

Throughout the last day the system was in operation (12/02/2002), it was used for web browsing.

```
Mon Dec 02 2002 08:37:00     1024 .a. -/-rwxrwxrwx 48       0        993-128-4
/WINNT/system32/config/SECURITY.LOG
```

The system is shut down for the last time at Mon Dec 03 2002 14:47:29.

```
Mon Dec 02 2002 14:46:52   439824 .a. -/-rwxrwxrwx 48       0        975-128-4
/WINNT/Media/Windows NT Logoff Sound.wav
Mon Dec 02 2002 14:47:29  1306624 mac -/-rwxrwxrwx 32       0        994-128-0
/WINNT/system32/config/SYSTEM.ALT
```

Recover Deleted Files

The fls tool was used to report the files that had been deleted, but that still retained file names or other meaningful associations.

```
#fls -d -l -f ntfs -p -r disk1.dd > disk1.fls.deleted

#fls -d -l -f fat32 -p -r disk2.dd > disk2.fls.deleted

#wc *fls.deleted
3767 64885 610313 disk1.fls.deleted
4682 79292 824199 disk2.fls.deleted
```

To determine whether any well known malware, such as network sniffers or hacker tools, had been on this system and then deleted, the list of deleted files was searched for the ".exe" string. If any suspicious .exe files were found, they could be undeleted using the dd method illustrated earlier in this paper. While an entire intact binary may not be recovered to a functional condition, the strings embedded in it might reveal its identity and/or purpose.

```
C:\PRACTICAL>grep –I "\.exe" disk1.fls.deleted >
exe.deleted

C:\PRACTICAL>grep –I "\.exe" disk2.fls.deleted >
ositis.deleted

C:\PRACTICAL>wc *.deleted
    142    2317   23253 exe.deleted
     11     184    1956 ositis.deleted
    153    2501   25209 total
```

So there were very few deleted files identified by fls, and none that stood out as red flags.

Below is a portion of the disk1.fls.delete report file created with fls above.

```
disk1.fls.deleted.files:r/r * 7093-128-4(realloc):
      Inoculan/WINTDIST.EXE   1997.06.01 13:30:12 (EDT)     1999.08.13
14:40:31 (EDT)    2001.09.22 12:17:48 (EDT)     63850 0     48

disk1.fls.deleted.files:r/d * 6939-144-6(realloc):
      Mobius/DDR/RDSWIN.EXE   2001.07.26 11:49:57 (EDT)     2001.12.03
08:44:29 (EST)    2001.07.26 11:49:57 (EDT)     56    0     48

disk1.fls.deleted.files:r/r * 9377-128-1(realloc):
      Mobius/DDR/VBLOCK.EXE   1998.04.08 03:16:22 (EDT)     1999.12.16
21:33:29 (EST)    2001.09.22 12:19:57 (EDT)     362   0     48

disk1.fls.deleted.files:r/r * 3774-128-1(realloc):
      Mobius/DDR/VDRCFG.EXE   2001.06.28 09:56:56 (EDT)
```

String Search

While there have been a few indications that this system has been used in violation of policy, there has been nothing to support a prolonged pattern of inappropriate activity. This is, until this point!

There are 48 pages of preparation preceding this point in the investigation. It may seem that an analyst could simply jump to this type of point, find the evidence and be done. However, the different data gathering and detailed reporting tasks were performed before any type of analysis for several reasons, including:

- To establish foundation that would support the integrity of the findings. Each of the little tasks, such as creating and comparing md5 hashes or documenting disk and file system characteristics make a statement either in support of against the items the analyst reports. This is a puzzle that demands completion. Any pieces left over or that do not quite fit invalidate or distort the picture the analyst provides.
- To methodically comb through the system without overlooking items that are obscure, but prove to be critical. Bad behavior is systemic rather than focused. While there may be solid evidence of one type of activity that was discovered through one task (inspecting the web browser history files), the same task may not identify other solid evidence normally discovered through another task (manually reviewing deleted word-processing documents that the analyst was able to undelete).
- Document the process in support of the final report. A forensic analysis usually takes several days, or weeks, to complete. The operative word is <u>complete</u>. Without detailed notes made throughout the process, facts will be forgotten and the analyst's rationale will be lost. When the time comes to write the final report, without detailed notes taken throughout the entire process, it will be extremely difficult to effectively report or rationally support the analyst's finding

During the string searching, evidence emerged that this system has been used to visit a large number of pornographic web sites over a relatively long period of time. While the determination of criminal activity is ambiguous (more on that at the end of this document), there is no question that this is a significant violation of policy that will be referred to the appropriate department for action. Previous cases of this type have led to dismissal, putting the seriousness of this investigation into concrete perspective for the first time.

To this point, the analysis tended to focus on the areas known to often hold clues or suggestions of where next to look. This has provided a detailed view of this system, along with many aspects of the operator's activities. Frequently, the strings and grep tools have been used in combination to ferret out any existing hints of inappropriate activity, though nothing of significance has been identified to this point.

The routine inspection of these well known locations is a very effective first step in analysis because:

- they are easy to check;
- they are likely to contain clues or evidence of the activities being investigated; and
- the average user is not sophisticated enough to sanitize these areas.

So, the entire disk image will be searched for the same types of keywords used before when analyzing individual files, or groups of similar files.

First the raw disk image is processed with the strings tool. Since this disk image is from a Win NT system, is assumed that a large portion of the content could be in Unicode format, along with ASCII. Since the Linux version of strings does not support Unicode, a Windows version was used. The first strings pass looked for Unicode, while the second used the –a parameter to change the default Unicode encoding to ASCII. Each pass redirected its results to separate files that were then tallied with wc.

An added advantage of running strings on the entire disk image, rather than just the files in the file system, is that many file systems support alternate data streams (ADS), which allows for the branching of a filename to include other unrelated files. There are several reliable tools available that can scan an entire file system and identify any ADSs. However, in this case Disk1 uses FAT, which does not support ADS, and Disk2 uses NTFS, which does support ADS. But the NTFS disk image is broken into to pieces that do not constitute a functional file system. Fortunately, the contents of an ADS file are just as visible to strings as the primary data stream (the stream that contains the contents of the given filename).

```
C:\PRACTICAL>strings disk1.dd > disk1.strings

C:\PRACTICAL>strings -a disk2.dd > disk1.a.strings
```

Rather than being subtle, a small list of unambiguous words that are often associated with malware (greets, 31337, hack) and adult web content (horny, slut) were fed into grep for searching through the strings culled from the raw disk image.

```
C:\PRACTICAL>cat hackwords.txt
greets
31337
hack
horny
slut

C:\PRACTICAL>wc *.strings
34680083 50425618 448478010 disk1.a.strings
5067489  6965337  65715866  disk1.strings
39747572 57390955 514193876 total

C:\PRACTICAL>grep –i –f hackwords.txt disk1.strings >
disk1.strings.of.interest

C:\PRACTICAL>grep -i -f hackwords.txt disk1.a.strings | wc
    450    5285   59257

C:\PRACTICAL>grep -i -f hackwords.txt disk1.strings | wc
    471    1230   15879
```

While the initial results contained lots of digital white noise (material that met the grep search criteria in letter, but not in spirit) it quickly became obvious that there were significant findings here.

After a quick review of the disk1.strings.of.interest file, it is obvious that this system had been used to browse a large number of adult web sites. While there was hundreds of these sites listed, each with clearly adult themes, none of them appeared in the IE logs.

But why was there no indication of this in the browser history logs? The answer was found in the Unicode strings. Most of the strings that included these keywords also included the string "Program Files\Ositis Software\WinProxy Lite\Cache\". This was a reference to a directory. After a quick check, this directory was located. It contained a web proxy that had filtered the inappropriate URLs before they could be recorded by the IE browser. (OSITIS SOFTWARE, WinProxy [47])

This local proxy program accounted for the absence any suspicious entries in the browser's logs, it was interesting that no other logging mechanism, error reporting, or the file system seemed to be aware of this local proxy software, or its activities. The only proof of the proxies' existence and operation is from the deleted .jpg files recovered by jpg-recovery.pl and the many, descriptively named files in its cache folder. The fact the proxy software itself had been deleted is possibly evidence itself. That sort of "leave no footprints" behavior does make sense, considering why this person appears to have used a local proxy.

The ls tool was used to create a report that clearly illustrates the contents of this folder sorted by their creation dates. This will show the time frame during which this activity occurred, along with a rough distribution of this activity.

```
#ls --time=ctime --sort=time -o --time-style=full >
/root/morgue/ositis.cache
```

The \Cache folder contains 238 log entries, with 131 referring to clearly pornographic web sites, and most of the others implied sexual explicit names that were not clearly such. These 131 log entries ranged in date from 2000-01-10 until 2001-07-10.

Rather than being subtle, a small list of unambiguous sexually explicit words was used to again search the raw disk image with the grep tool. These proxy cache items were clearly smoke, but the location and characteristics of the fire still had to be pinpointed.

Working on the assumption that at least some of these image files had existed on this disk, the jpg-recover.pl script was used to read through the disk1.dd image file and attempt to recover every .jpg picture file that could be found.

This script had been found on the Internet while searching for "forensic script undelete." Jpg-recovery.pl had been coded to recover .jpg files from digital camera memory sticks.

After about 12 hours (at 99% CPU usage), the script reported identifying and recovering 83 jpg files, none were of interest. After 24 hours, 300 more images had been recovered, a few of which were hardcore pornography and seemed to match the topics implied in the URLs. After about 48 hours, jpg-recovery.pl had processed the entire disk image and recovered 10,461 .jpg files. A few randomly selected samples were viewed while the process ran. The majority turned out to be pornographic.

While the majority were pornographic and clearly in violation of policy, a large number were the ancillary images that are scattered over most web pages; navigation buttons, ads, and logos. In may cases these otherwise benign images were themselves pornographic. A large number of the images were thumbnails of larger versions. Often the larger version of the image was listed soon after the thumbnail version. This indicates the person was actively selecting specific images (thumbnails) and downloading the larger version. There are instances where a person can stumble onto a pornographic site that immediately displays explicit images, but these images are almost always small thumbnail versions. When the person clicks to leave the site, a cascade of other pornographic sites begin popping up on the screen completely beyond the person's control. This would account for a noticeable number of pornographic thumbnail images, but not for an equally large number of full sized image files. This shows the is not a fluke, but a deliberate and sustained behavior.

Below is a portion of the display while jpg-recovery.pl is running. Notice that it reports its progress by the byte offset it is passing. Especially useful is the byte offsets of the start of an identified .jpg and the number of bytes considered to make up the file.

```
# perl jpg-recovery.pl
Scanning 'disk2.dd' looking for pictures...
Passing byte 102400 of disk2.dd ...
Passing byte 204800 of disk2.dd ...
Passing byte 307200 of disk2.dd ...
(lines omitted)
Passing byte 37785600 of disk2.dd ...
Image #2: start found, dumping to lostpic002.jpg ...
Passing byte 37888000 of disk2.dd ...
Passing byte 37990400 of disk2.dd ...
Done -- wrote ~170986 bytes to lostpic002.jpg (JPEG end @ 38031339)
Passing byte 38092800 of disk2.dd ...
(lines omitted)
Passing byte 1941606400 of disk2.dd ...
Passing byte 1941708800 of disk2.dd ...
Passing byte 1941811200 of disk2.dd ...
Done -- wrote ~3638272 bytes to lostpic10461.jpg (EOF @ 1941827584.)
Finished scanning disk2.dd.

Created 10461 file(s) -- check 'em out!

# ls lost* -o
-rw-r--r--    1 root       370909 Apr 21 14:42 lostpic001.jpg
-rw-r--r--    1 root       170987 Apr 21 14:42 lostpic002.jpg
(lines omitted)
```

The rdjpgcom [(49)] was used to get both the comments and the jpg image dimensions. Room is reserved within each .jpg file for comments, usually provided by the software creating the .jpg. This can help identify the software used to create or modify an image and, in some cases, is used by an intruder to "brand" the image as theirs.

```
# rdjpgcom -verbose lostpic001.jpg

APP12 contains:
Ducky\000\001\000\004\000\000\000\026\000\000
JPEG image is 86w * 57h, 3 color components, 8 bits per sample
JPEG process: Baseline

# rdjpgcom -verbose lostpic002.jpg
Adobe ImageReady
APP12 contains:
Ducky\000\001\000\004\000\000\0001\000\000
JPEG image is 65w * 65h, 3 color components, 8 bits per sample
JPEG process: Baseline
```

The dd tool can be used to manually salvage a single file. Using lostpic002.jpg as an example, the starting byte offset and the byte count of the file is all that dd needs in order to copy that portion out of the raw disk image to a .jpg picture file.

```
      endpoint – size = startpoint

      38031339 – 170986 + 1 = 37860353
```

The +1 simply align the first and last bytes from the perspective of the dd program.

Here the lostpic002.jpg that jpg-recovery.pl identified and recovered is likewise recovered by manually using dd. Not to be too simplistic, the process that jpg-recovery.pl or the manual dd methods use is to search the disk image file for the header information that identifies that start of a .jpg formatted file, which is “JIFF.” There are then several possible binary strings that represent the end of the file.

Having located the start, you would harvest everything until either identifying one of the terminating signatures, or you had reached a volume beyond what you would expect the file size to be. Knowing then the start and end byte offsets, you simply copy them to a new file with a .jpg suffix.

```
# dd if=disk2.dd of=pic2.jpg bs=1 skip=37860352 count=170987

# ls *2.jpg -o
-rw-r--r--    1 root       170987 Apr 21 14:42 lostpic001.jpg
-rw-r--r--    1 root       170987 Apr 21 19:58 pic2.jpg

# md5sum lostpic002.jpg pic2.jpg
32decd4e1ca88a19220c58fd0b97753a lostpic002.jpg
32decd4e1ca88a19220c58fd0b97753a pic2.jpg
```

After using dd to pull out the .jpg file, its byte count and md5 hash are compared to the version jpg-recovery.pl found, validating the process. Of course, both image files are viewed to determine if they are meaningful or damaged.

```
C:\PRACTICAL> grep –I "Ositis Software" disk1.unicode.strings >
disk1.grep.Ositis
```

```
C:\PRACTICAL>wc 64.byte.ascii
   2901    2901  305513 64.byte.ascii
```

This file was then opened in a text editor where several "clutter" strings were removed using search-and-replace, all characters were changed to uppercase. This was to begin shaping these findings into a list of URLs. It was hoped that the basic statistics of these URLs would be a clue themselves.

The .jpg files that had been recovered did not include MAC time information, so these could not be correlated with the proxy logs. It is possible that other file recover tools could have been used that would have included data from the file system metadata files such as the FAT or super block.

Having this raw data, the pornographic URLs, shows what the activity was, but that activity needs to be put into meaningful context. In a murder case, the prosecutor does not focus on the fact a murder occurred (this is the *what* question), but rather on presenting the timeline of events, motive, and other items that address the *who*, *when*, *why*, and *how* questions. The *what* question (a murder) is not the focus. Answers to the *who, what, when, why,* and *how* questions will support the *who* premise.

First the raw data is captured for analysis. The fact these are clearly pornographic web sites is not the focus. Instead, the characteristics of these items will provide a context that is just as important.

All of the proxy cache files are captured into a file (disk1.Ositis).

```
C:\PRACTICAL> grep -i "Ositis Software" disk1.unicode.strings > disk1.Ositis
```

After inspecting a few of these cache files, it is clear that each can be treated as a variable length logical record. The cut tool is used to parse out the pertinent fields to an extraction file for further manipulation. In this example, the cut tool is instructed to use the hyphen (-d “-“) as the field delimiter. The fields to be parsed out are fields 3 through 50 (-f 3-50).

This data is piped into the sort program, and from the sort program into a results file (disk1.ositis.sorted).

```
C:\PRACTICAL> cut -d "-" -f 3-50 disk1.Ositis | sort > disk1.ositis.sorted
```

Now that the data is organized in this way, each line was then tallied, using uniq. After sorting the output file of the command below, the format listed URLs by their visit frequency.

```
C:\PRACTICAL> uniq -c disk1.ositis.sorted > disk1.URLs.count
```

At this point, there were:

- 14,599 URLs (showing the volume of surfing);
- 1,148 unique (showing the range of sites visited);
- 348 sites had been visited 15 or more times; and
- 80 of the visited sites were clearly pornographic.

Steganography

Being "inappropriate," these images are pertinent to the study. But, their content (the picture they portray) is not the end of the analysis. One more step is necessary.

Taking into account the apparent source of these images (adult web sites), they may contain material that has been embedded using steganography (hidden writing) techniques. Until 9/11, steganography was widely understood in the underground, but not in the general public. Since the technique of "hiding" material within an image file without altering its perceptible content was seen as "secure" since it was not perceptible, it is arguable that weaker passwords were used.

The subject of steganography is widely and well documented, especially in the SANS Reading Room [32], so I will not take time here to describe steganography in any detail. Suffice it to say that steganography is a family of techniques for injecting new material or replacing exiting portions of a file with new material, typically an image format file, in such a way that the perception and functionality of the original is not perceptible altered and so the new material can be retrieved later. A good analogy that is often used is to say encryption is putting a message in an envelope and marking the envelope ***secret,*** *w*hile steganography is mixing the message into the text on a postcard. An observer would instantly know that the envelope contained a secret message, but would not notice anything unusual or enticing about the postcard, though both contain the same sensitive message.

In the example below, the stegdetect [30] tool is run to determine the probability that an image has been altered. In this example, the Windows version of stegdetect was run, though the Linux version tends to be faster on the same system.

In the following example, the parameters "tjpoi" are used. The t parameter sets the *tests* to be performed to jpoi. These letters each represent a distinct tool available on the Internet for embedding material into .jpg images, specifically **o**utguess, j**p**hide, and **j**steg-shell.

The "s1.000" parameter sets the sensitivity of the analysis; the higher the number the more sensitive and more prone to false positives. When run with the default value of 1.000, stegdetect identifies nine images as having been tampered with.

To reduce the white noise, the stegdetect results are piped to grep in order to filter out all of the negatives and display only those images suspected of tampering. By reducing the sensitivity to one-half (0.500), the borderline or questionable positives are filtered out, leaving only the most likely positives.

After some trial and error, a sensitivity setting of 0.250 produced only 2 strong positives. These two will be subjected to brute force password cracking using the stegbreak [31] tool that is bundled with stegdetect [30].

Before starting the stegbreak session, the subject .jpg files were converted (using the –c parameter as seen below) into a much smaller representation that contained all of the material stegbreak needed to attempt to guess the passwords. In effect, the image material was removed leaving only the authentication (password) framework that stegbreak would attempt to crack.

```
# ./stegbreak -c /root/morgue/*.jpg

/root/morgue/lostpic926.jpg: converted to .jph
/root/morgue/lostpic932.jpg: converted to .jph
Converted 2 files.
```

The same wordlist that was used earlier in john-the-ripper is reused here. In fact, the configuration file for john-the-ripper that fine tunes its operation is also used verbatim (which is a nice feature of stegbreak).

```
# wc bfwordlist.txt
1402823  1481730 14913 986  bfwordlist.txt

# ./stegbreak –t p –r rules.ini –f bfwordlist.txt /root/morgue/*.jph

Loaded 2 files…
Status:   0.000%,    761.1 c/s: 0bumb1247i0n
Status:  28.947%,    346.7 c/s: koppel4
Status:  36.298%,    324.8 c/s: platterful.
/root/morgue/lostpic926.jpg : negative
/root/morgue/lostpic932.jpg : negative
Processed 2 files, found 0 embeddings.
Time: 251888 seconds: Cracks: 83229784,  330.4 c/s
```

Running a 330.4 crack attempts per second, stegbreak was unable to guess the password after 2.9 days nonstop, or approximately 83,223,795 attempts. While on the subject of encryption, the raw disk image file was checked for blocks of ASCII armored material.

There are no indications of encryption programs, at least by file name. None of the existing files end with a common encryption suffix such as .pgp or .gpg. Indications of encryption programs might take the form of chunks of encrypted (large groups of consistently random characters) and/or ASCII armored material on the disk. While ASCII armoring is encoding, not encryption, it is often used with encrypted for transmitting binary material over communications systems that do not support 8-bit bytes.

ASCII armoring is the process of converting binary material, which likely contains nonprintable characters, into a material that is equivalent, though composed entirely of the standard printable characters. One common use of ASCII armoring is to prepare a binary file for transmission across a communications network where a nonprintable character(s) in the binary might be interpreted as a command-bearing control character causing the communications components to take unintended action.

In some cases, the character space of a binary may not be supported by the transmission medium. ASCII encoding has a range of 256 possible characters (0-254, or 2^8), each represented with an eight-bit byte. Standard, universally supported ASCII makes use of the first seven significant bits of the eight. This gives a minimum of 127 characters (2^7). By making use of the eighth bit, an additional 129 characters can be represented. For the most part, it is these additional 129 characters that some communications systems do not support. So ASCII armoring re-encodes the material that has a 2^8 character set to a character set of 2^7

To get an overview of the presence of any ASCII armored files or artifacts, the strings and grep tools were again used.

```
C:\>strings -a -n 64 disk1.dd | grep -v -b "\ " > 64.byte.ascii

C:\>strings -n 64 disk1.dd | grep -v -b "\ " > 64.byte.Unicode
```

Two common characteristics of ASCII-armored material are a fixed-line length of 64 bytes and the absence of spaces.

The strings command would simply identify strings that were at least 64 characters long and passes them to the grep tool. Grep inspects each line and drops those containing even one space character. These final lines are then redirected to the 64.byte.ascii file for review.

```
C:\practica>wc 64.byte*
 291629  291627 26051224 64.byte.ascii
 166954 1266451 16301542 64.byte.unicode
 458583 1558078 42352766 total
```

After reviewing these results, did not appear that any meaningful use of ASCII armoring was found on this disk.

Conclusions

This project began with three specific goals. To conclude the answers to each question will be summarized.

1. Is confidential or proprietary information being unintentionally leaked through the surplus of system?

Regarding this one sample system, no confidential or proprietary information was identified. However, there appears to have been no steps taken to prevent the disclosure of confidential, proprietary, or any other type of information. Documents, email, and applications dating back several years were recovered demonstrating a failure to follow relevant policies or procedures and the likelihood of sensitive material unknowingly being released.

2. Do these systems show signs of having been compromised by an unauthorized intruder, or of previously undetected malware (malicious software, viruses, or adware)?

This one sample system had indications of both virus attacks and system crashes. But there was no indication of the actual presence of malware.

3. And, do these systems show evidence of inappropriate use.

There is clear evidence this host has was used for approximately 18 months for extensive browsing of pornographic web sites. In addition, an unauthorized local proxy program had been installed (also against policy and procedure) and used in an attempt to hide or obscure this activity.

This also indicates a failure on the part of network administration to recognize a large and consistent volume of clearly identifiable activity. This increased the risk of exposure due negligently allowing such activity, and by the attack vector this created for the malware pervasive through such sites.

The failure to prevent or detect the installation of a local proxy program is another significant finding.

To conclude, the results of this project brings to light several highly significant findings of risk both to the technical infrastructure and company exposure to legal action. Remediation can be addresses by

1. Revising policy and procedures to address these findings,
2. informing staff of the policies, and educating staff on how to adhered to them,
3. regular validation of the effectiveness of these policy and procedures, and by
4. consistently enforcing the consequences of violation.

Part 3 - Legal Issues of Incident Handling

The scenario

> You are the system administrator for an Internet Service Provider that provides Internet access to paying customers. You receive a telephone call from a law enforcement officer who informs you that an account on your system was used to hack into a government computer. He asks you to verify the activity by reviewing your logs and determine if your logs reflect whether or not the activity was initiated there or from another upstream provider. You review your logs and can only determine a valid user account logged in via a dialup account during the period of the suspicious activity.
>
> NOTE: For the purposes of this scenario, assume you validated the identity of the law enforcement officer and this is not social engineering.

This being a rather ambiguous scenario with more being implied than stated, several assumptions will first be made in order to better put the answers into context.

- As a system administrator I am competent to perform all of the technical tasks involved in this situation.
- As system administrator, while I have had some training regarding the legal requirements involved with technology, at each step I will have consulted with the appropriate legal counsel.
- "An account on my system" refers to a user account that I, as an ISP, provided to a customer. This rather than a local user account on my local host system.
- The attacking account is known by at least an IP address, and possibly a DNS name. Having only **Hostname=joe** and **domain=ACME.ISP.NET** seems too vague for law enforcement to actually begin requesting information.

There are several pieces of Federal law which address information technology. When assessing legal possible implications of a situation, one or more of these will likely apply to some degree. Of course, state and local law may also apply. These include:

- The Fourth Amendment addresses the individual's reasonable expectation of privacy. (54 DOJ I.b)
- The Privacy Protection Act (PPA) focuses on freedom of speech as presented in the First Amendment. (54 DOJ II.B.2.a)
- The Electronic Communications Privacy Act (ECPA) lays out protection and process regarding digitally stored communications. (54 DOJ III)
- The Wiretap Statute, a.k.a. Title III addresses the privacy of real-time communications. (54 DOJ IV.D)
- The Computer Fraud and Abuse Act (CFAA) 18 U.S.C. § 1030 (54) focuses on unauthorized access to computer systems. (56 Robinson)

Questions

A. What, if any, information can you provide to the law enforcement officer over the phone during the initial contact?

The officer's question is *do my logs reflect the activity in question*. It is easy to treat it as an open ended question that includes "Who was logged, and as doing what?" But from a legal perspective, the jump from "did this occur?" to the details of the identity of the doer and the content of that communications is huge. It sounds simple, but it is critical that only the questions actually being asked are addressed, whither the answer is yes, no, or "that information requires a request from a judge or court (subpoena, warrant, or court order)."

After a quick and objective review of my logs (while the officer is on the phone), I can report back that there does not appear to be any indication of "activity" from upstream accounts during this time period, and that a "valid user account" was connected at the time the government system was hacked.

At this point I may begin to think the logs identify a hacker who law enforcement is investigating, but I have to keep in mind that there is significant legal protection of this person's rights to privacy, specifically regarding their personal information (SSN, address, bank account), their real-time communications, and their stored communications.

While ECPA begins with the principal that a provider can release no information, there are several exceptions that would allow me as a provider to volunteer information about a subscriber or their communications. However, none of these exceptions seem to apply to this situation. (54 DOJ III.E) So, while at this point I might provide information about the operation of the system (there were subscribers connected, but no upstream connections), I cannot provide information about the subscribers.

B. What must the law enforcement officer do to ensure you to preserve this evidence if there is a delay in obtaining any required legal authority?

The preservation of evidence is addressed under 18 U.S.C. § 2703(f). § 2703(f)(1) specifically states (54 DOJ III.G.1):

> A provider of wire or electronic communication service or a remote computing service, upon the request of a governmental entity, shall take all necessary steps to preserve records and other evidence in its possession pending the issuance of a court order or other process.

This phone call alone is presumably enough to start the process. The officer can verbally request that the exiting pertinent logs (not future logs) be "frozen" so that they will be available in the near future (within 90 days) should additional actions such as subpoenas, court orders, or search warrants be issued, 18 U.S.C. § 2703(f)(2). During this 90-day period, the officer must acquire and present an order (subpoena, warrant, court order) that supports this request. (54 DOJ III.G.1)

This is a practical adjustment to the fast paced realities of cyberspace. In the time it could take to properly acquire a court order, the situation is in dynamic flux, and the majority of any tangible evidence of a crime may have a 12-hour life span after which it is overwritten, similar to surveillance cameras that retain only the last few hours. In order to avoid investing in hundreds or thousands of videotapes, the assumption is made that a few tapes can be reused in a regular cycle and that very soon after an incident; the tape will be pulled and secured as evidence.

Regardless, I would request that the officer fax/email this request to my office as soon as possible in order to document the request and to avoid miscommunications.

C. What legal authority, if any, does the law enforcement officer need to provide to you in order for you to send him your logs?

If the logs did in fact clearly provide concrete information relevant to the officer's request, I could choose to provide the "basic" information.

However, I would insist that the officer first acquire a subpoena for the specific logs identified by date and time (GMT), source and destination IP address and TCP port numbers (if any).

When it came time to produce this data, I would extract only the log records that meet these criteria. I would then parse out only the fields that were indicated in the subpoena. The point here is to make sure to include only the basic information that a subpoena has the power to illicit in accordance with ECPA.

D. What other "investigative" activity are you permitted to conduct at this time?

The Wiretap Act, Title III 18 U.S.C § 2510-2522, addresses the real-time surveillance (wire-tap or sniffing) of electronic communications, including the content of the communication. While Title III prohibits a person from monitoring the content of electronic communications, or for law enforcement officials to accept the fruits of that surveillance, the provider exception of Title III, 18 U.S.C § 2511(2)(a)(i) does specifically allow such surveillance and disclosure to law enforcement when preformed on their own network by the owning service provider.

> Providers investigating unauthorized use of their systems have broad authority to monitor and then disclose evidence of unauthorized use under § 2511(2)(a)(i) but should attempt to tailor their monitoring and disclosure so as to minimize the interception and disclosure of private communications unrelated to the investigation. (54 DOJ IV.D.1.3.c)

More specifically,

> “protection of the rights or property of the provider” clause of § 2511(2)(a)(i) grants providers the right “to intercept and monitor [communications] placed over their facilities in order to combat fraud and theft of services.” (54 DOJ IV.D.3.c)
>
> Because monitoring a hacker's attack ordinarily does not violate the hacker's reasonable expectation of privacy, see "Constitutional Suppression Remedies," infra, it is unclear whether a hacker can be an "aggrieved person" entitled to move for suppression of such monitoring under § 2518(10)(a). No court has addressed this question directly. Of course, civil and criminal penalties for unlawful monitoring continue to exist, even if the unlawful monitoring itself targets unauthorized use. (54 DOJ E.1.a)

Relying on the provider exception clause, I would begin additional logging and monitoring in accordance with company procedures relating to “suspicious activity”. That procedure states “when there is reason to believe that malicious, destructive, suspicious, or otherwise unauthorized activity is or is about to occur, steps to identify and contain that activity should be taken in order to protect the integrity of the systems and the confidentiality of the customers.” This in-house procedure also demonstrates that such actions are “normal” in the operation of this system. (54 DOJ IV.3.D.3)

In addition, The Trace Devices chapter of Title 18, 18 U.S.C. § 3121-3127 address real-time electronic surveillance, network sniffing in this case. §3121(b) specifically allows me to pen/trap (sniffing both inbound and outbound traffic) my own network without a court order. (54 DOJ IV.c)

E. How would your actions change if your logs disclosed a hacker gained unauthorized access to your system at some point, created an account for him/her to use, and used THAT account to hack into the government system?

After updating my resume, I would immediately secure the evidence (logs, files, screen shots) that lead me to this discovery. This evidence would be copied to non-volatile media, such as CDROM, and each digital file would be hashed with an md5 utility. Copies of the hash values would be stored with the evidence. The hash values would also be used to verify the integrity of the copies, demonstrating that the copies on CDROM are identical to the originals, and that there have been no subsequent changes throughout the life of this investigation.

The Best Evidence Rule, Fed. R. Evid. 1001(3) states that “[i]f data are stored in a computer or similar device, any printout or other output readable by sight, shown to reflect the data accurately, is an “original”.” (54 DOJ V.D.1) The hash values should satisfy the “shown to reflect the data accurately” requirement. In fact, hashing each piece of digital material involved in an investigation is a good practice.

So according to the Best Evidence Rule, since the files I have copied are “readable by sight”, and can be “shown to reflect the data accurately” using the md5 hash values, these files can be considered “original”.

I would then contact the law enforcement officer who first called, on the assumption they would have the best understanding of the situation, and inform them of my findings and offer to provide a copy of my findings.

At this point, the Computer Trespasser Exception, 18 U.S.C. § 2511(2)(i) comes into play.

> The computer trespasser exception may be used in combination with other authorities, such as the provider exception of § 2511(2)(a)(i). A provider who has monitored its system to protect its rights and property under § 2511(2)(a)(i), and who has subsequently contacted law enforcement to report some criminal activity, may continue to monitor the criminal activity on its system under the direction of law enforcement using the computer trespasser exception. In such circumstances, the provider will then be acting under color of law as an agent of the government. (54 DOJ IV.D.3.d)

I would continue monitoring and logging, making sure to capture that information to non-volatile media so that the intruder could not alter or destroy it. Depending on what alerted me to the intruder's presence and activities I would direct increased logging and monitoring at that area.

The Fourth Amendment provides for the reasonable expectation of privacy. Even though I am convinced that this intruder is "guilty", I cannot haphazardly violate their constitutional rights, including their right to privacy while they exercise unauthorized access (in my opinion) to a ("my") protected computer.

Since this intruder has demonstrated highly privileged access to the system by creating an account for their use, I have to assume they have highly privileged control of the logging process and files. I also have to assume there is a quickly escalating probability the intruder will realize they have been discovered and take steps to cover their track and destroy evidence of their activities.

Regarding ECPA, there is a provision for the voluntary disclosure of content material (not just the basic material) when disclosure "may be necessarily incident to the rendition of the service or the to protection of the rights and property of the provider of that service," § 2702(b)(5). In addition, ECPA allows for my voluntary disclosure of basic information to a governmental agency when disclosure "may be necessarily incident to the rendition of the service or the to protection of the rights and property of the provider of that service," § 2702(c)(3).

At this point, state law enters the picture. Kentucky Revised Statue 434.845-860 addresses the unlawful access to a computer.

> 434.845 Unlawful access to a computer in the first degree.
> (1) A person is guilty of unlawful access to a computer in the first degree when he or she, without the effective consent of the owner, knowingly and willfully,

> directly or indirectly accesses, causes to be accessed, or attempts to access any computer software, computer program, data, computer, computer system, computer network, or any part thereof, for the purpose of:
> (a) Devising or executing any scheme or artifice to defraud; or
> (b) Obtaining money, property, or services for themselves or another by means of false or fraudulent pretenses, representations, or promises.
> (2) Unlawful access to a computer in the first degree is a Class C felony.
> Effective: July 15, 2002
> History: Amended 2002 Ky. Acts ch. 350, sec. 2, effective July 15, 2002. -- Created 1984 Ky. Acts ch. 210, sec. 2, effective July 13, 1984. (53 KRS)

So in addition to federal law, the intruder can also be charge for the unlawful access to a computer with the intention of obtaining services. However, state law does not address the issues of intercepting communications (either real time or stored) leaving all other matters to federal law.

References

In some cases, the description is that provided by the page itself.

9. Redhat 8.0
   URL: http://www.redhat.com (3/12/03)
10. Microsoft XP
    URL: http://www.microsoft.com/windowsxp/default.asp (4/10/03)
11. ext2fs V0.10 – Extended File System version 2
    URL: http://ext2.yeah.net (4/10/03)
12. kernel-ntfs-2.4.18-14.i686.rpm
    URL: http://linux-ntfs.sourceforge.net/info/redhat.html (4/10/03)
13. GNU utilities for Win32
    URL: http://unxutils.sourceforge.net/ (4/19/03)
14. BreakPoint Software - The Home of The Hex Workshop Hex Editor
    URL: http://www.bpsoft.com/ (4/10/03)
15. Nmap ("Network Mapper") is an open source utility for network exploration or security auditing.
    URL: http://www.insecure.org/nmap/ (4/19/03)
16. Tcpdump
    URL: http://www.tcpdump.org/ (4/19/03)
17. Trinux is a ramdisk-based Linux distribution that boots from a single floppy or CD-ROM, loads it packages from an HTTP/FTP server, a FAT/NTFS/ISO file system, or additional floppies.
    URL: http://trinux.sourceforge.net/ (4/10/03)
18. Unzip
    URL: http://www.free-screensavers.de/e_unzip.htm (4/19/03(
19. Md5sum. Message Digest version 5, bundled with Redhat (1) and the GNU utilities for Win32 (5)
20. Arne Vidstrom's MACMatch
    URL: http://ntsecurity.nu/toolbox/ (4/19/03)
21. Unzip (linux) Included with Redhat Linux (1)
22. Zipinfo (linux) Included with Redhat Linux (1)
23. showacls Included with the NT Resource Kit (NTRK)
    URL: http://www.microsoft.com/ntserver/nts/downloads/recommended/ntkit/default.asp (4/19/03)
24. File tool. Included with Redhat Linux (1)
25. Stat tool. Included with Redhat Linux (1)
26. Ldd tool. Included with Redhat Linux (1)
27. Strings for Win32 by Mark Russinovich
    URL: http://www.sysinternals.com/ntw2k/source/misc.shtml#Strings (4/19/03)
28. Phrack Magazine, Volume 7, Issue 49, Article 06 of 16, "Project Loki" (3/10/03)
    URL: http://www.phrack.com/show.php?p=49&a=6

    Phrack Magazine   Volume 7, Issue 51 September 01, 1997, article 06 of 17, "L O K I 2 (the implementation)" (3/10/03)
    URL: http://www.phrack.org/show.php?p=51&a=06

    X-Force loki (1452)
    URL: http://www.iss.net/security_center/static/1452.php and http://secinf.net/unix_security/LOKI2__informationtunneling_program_and_description.html (3/10/03)

29. fprintf() - From Keyword Search the Single UNIX Specification
URL: http://www.opengroup.org/onlinepubs/007908799/ (4/19/03)
30. strace
URL: http://www.liacs.nl/~wichert/strace/ (3/10/03)
31. Request for Comments (RFC): 792 Internet Control Message Protocol (ICMP)
URL: http://asg.web.cmu.edu/rfc/rfc792.html (4/19/03)
32. Busybox. BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc.
URL: http://www.busybox.net/ (4/19/03)
33. Gpg (linux) The GNU Privacy Guard. GnuPG is a complete and free replacement for PGP.
URL: http://www.gnupg.org/ (4/19/03)
34. Wipe (linux) Wipe is a secure file wiping utility.
URL: http://wipe.sourceforge.net/ (4/19/03)
35. Nc (linux and windows). Netcat has been dubbed the network Swiss army knife. It is a simple Unix utility that reads and writes data across network connections, using TCP or UDP protocol.
URL: http://www.atstake.com/research/tools/network_utilities/ (4/19/03)
36. dd (linux and windows) Included with Redhat Linux (1). A Windows version is included in the GNU utilities for Win32 (5).
37. HTCIA – High Technology Crime Investigation Association. September 2002, volume 3, issue 2, page 7. Contributed by Randal Shane, (rshane@ureach.com)
URL: www.htcia.org (2/27/03)
38. Stegdetect 4.0 Stegdetect is an automated tool for detecting steganographic content in images.
URL: http://www.outguess.org/detection.php (4/19/03)
39. Stegbreak is used to launch dictionary attacks against JSteg-Shell, JPHide and OutGuess 0.13b
URL: http://www.outguess.org/detection.php (4/19/03)
40. SANS Reading Room - Steganography.
URL: http://www.sans.org/rr/steg/ (2/16/03)
41. Request for Comments: 1321 - The MD5 Message-Digest Algorithm
URL: http://www.faqs.org/rfcs/rfc1321.html (4/19/03)
42. DBAN; "Darik's Boot and Nuke"
URL: http://dban.sourceforge.net/ (4/05/03)
43. IP_HDRINCL system function call
URL: http://unixhelp.ed.ac.uk/CGI/man-cgi?raw+7 (4/3/03)
44. noop slide
URL: http://project.honeynet.org/scans/scan20/scan20.html (4/3/03)
45. Daemon: *Abbreviation for* disk and execution monitor. A procedure that is invoked without being called explicitly whenever an alteration, an addition, or a deletion or other event occurs. [ANSDIT] *Pronounced* dee' mun.
URL: http://www.its.bldrdoc.gov/projects/devglossary/_daemon.html (4/3/03)
46. GNU gcc 2.7.2.1 compiler
2.7.2.1June 29, 1996 per http://gcc.gnu.org/releases.html
copy downloaded from http://ftp.gnu.org/old-gnu/gcc/ dated 1996-08-1407:00:00 AM
URL: http://abel.hive.no/linux/ib/ (4/19/03)
47. ld-linux.so.1
URL: http://rpmfind.net/linux/RPM/redhat/6.2/i386/ld.so-1.9.5-13.i386.html (4/3/03)

48.libc.so.5
URL: http://rpmfind.net//linux/RPM/contrib/libc6/i386/libc5-5.4.38-3.i386.html (4/19/03)

49.ping
URL: http://docencia.ac.upc.es/FIB/STD/misc/ping.html (4/19/03)

50.fsstat
URL: http://www.mcsr.olemiss.edu/cgi-bin/man-cgi?fsstat+1 (4/19/03)

51.IEHist dumps Internet Explorer history from index.dat files into delimited files suitable for import into other tools.
URL: http://www.cqure.net/tools.jsp?id=13 (4/19/03)

52.John the Ripper is a fast password cracker, Besides several crypt(3) password hash types most commonly found on various Unix flavors, supported out of the box are Kerberos AFS and Windows NT/2000/XP LM hashes...
URL: http://www.openwall.com/john/ (4/19/3)

53.SAMDump 1.04. Created by Dmitry Andrianov
URL: http://www.l0pht.com/l0phtcrack/dist/samdump.zip (12/4/02)

54.Microsoft's Really Hidden Files: A New Look at Forensics. Version 2.6b by The Riddler
URL: ther1ddler@fuckMicrosoft.com (3/1/03)

55.Ositsis WinProxy Lite
URL: http://www.winproxy.com/english/home/hm_business_home_en.asp (4/3/3)

56.fls, part of the TASK toolset.
URL: http://www.atstake.com/research/tools/task/ (12/4/02)

57.rdjpgcom - display text comments from a JPEG file
URL: http://www.engr.utk.edu/eccsw/jpeg/rdjpgcom.html (4/3/3)

58.The Autopsy Forensic Browser
URL: http://www.atstake.com/research/tools/forensic/ (4/3/3)

59.Ethereal – Network Protocol Analyzer
Version 0.9.2 © 1998-2000 Gerald Combs Gerald@etheral.com
URL: http://www.ethereal.com (4/3/3)

60.Richard Ginski – SANS GCFA practical
URL: http://www.giac.org/practical/GCFA/Richard_Ginski_GCFA.pdf (3/24/03)

61.Kentucky Revised Statutes Chapter 434 – Unlawful Access to a Computer in the first degree.
URL: http://www.lrc.state.ky.us/KRS/434-00/CHAPTER.HTM (3/24/03)

62. Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations – Computer Crime and Intellectual Property Section, Criminal Division, United States Department of Justice. July 2002
URL: http://www.cybercrime.gov/s&smanual2002.htm (3/1/03)

63. U.S. Information Security Law, Part one: Protecting Private Sector Systems, and Information Security Professionals and Trade Secrets. Steven Robinson
http://www.securityfocus.com/infocus/1669 (04/2/03)

64.U.S. Information Security Law, Part two: Protecting Private Sector Systems and Securing the Working Environment. Steven Robinson.
URL: http://www.securityfocus.com/infocus/1681 (4/2/03)

65.Issue 13: Kernel Korner: The ELF Object File Format by Dissection. Posted on Monday, May 01, 1995 by Eric Youngdale
URL: http://www.linuxjournal.com/article.php?sid=1060 (4/6/03)

66.NetWinder ELF Design Notes Pat Bierne, patb@corel.com Revision: 1.3, Date: 1999/12/07 16:24:04
URL: http://www.netwinder.org/~scottb/notes/Elf-Design-all.html

67. Symantec - Norton Antivirus
URL: http://www.symantec.com/ (3/1/03)
68. McAfee Antivirus
URL: http://www.mcafee.com (3/1/03)
69. Loki2 source code.
URL: http://www.hackerproof.org/technotes/backdoors/loki2.tar.gz (2/18/03)

Appendix A.
Sanitizing DASD

Statement of the problem

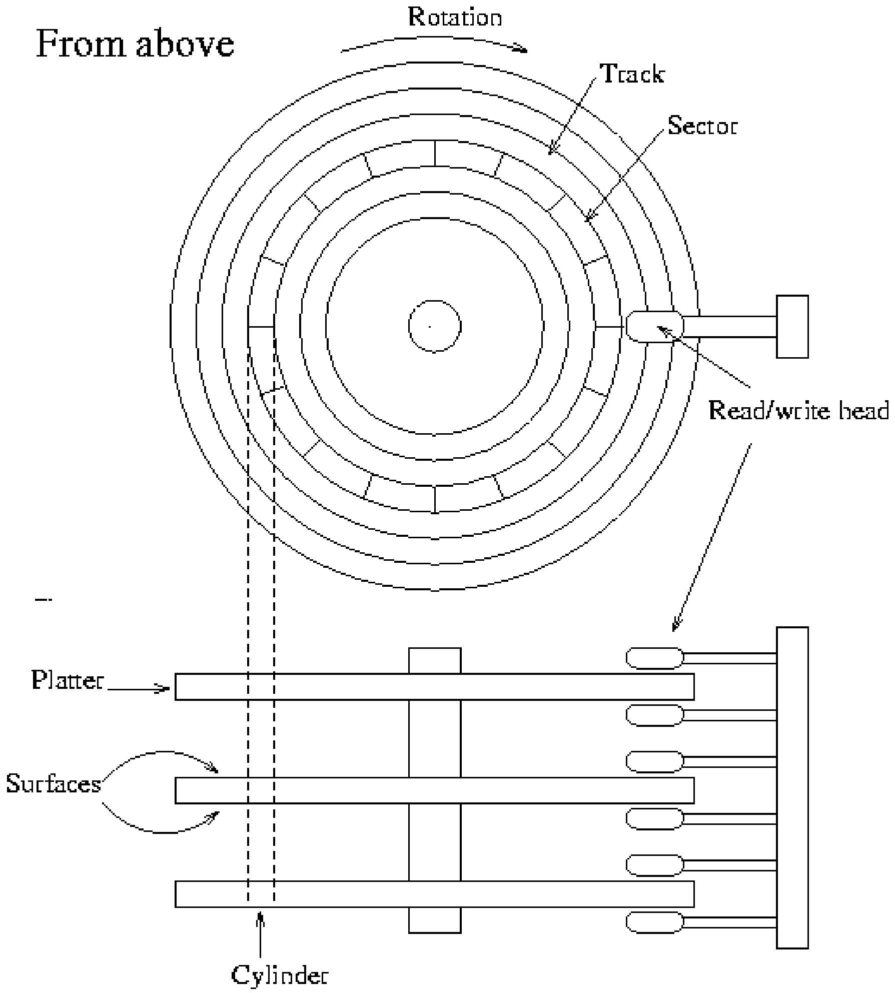
When a computer hard disk, or DASD (Direct Access Storage Device), is sent to surplus and then acquired by other agencies and the general public, it likely contains confidential, personal, or proprietary data and information[1].

This paper deals only with sanitizing hard disk drives. The architecture of floppy (5.25 or 3.5) diskettes is such that overwriting is not as effective as with hard disks. Since you cannot be sure of the history of a floppy (what types of material it has held), all floppy diskettes should be physically destroyed when no longer needed. Since CDROM disks cannot be effectively overwritten or erased, and scratching or marring the surface does not alter the still recoverable material under the surface, all CDROM disks should be physically destroyed when no longer needed.

Before addressing the effective deletion, or wiping, a hard disk, the basic physical and logical structure of a hard disk must be understood.

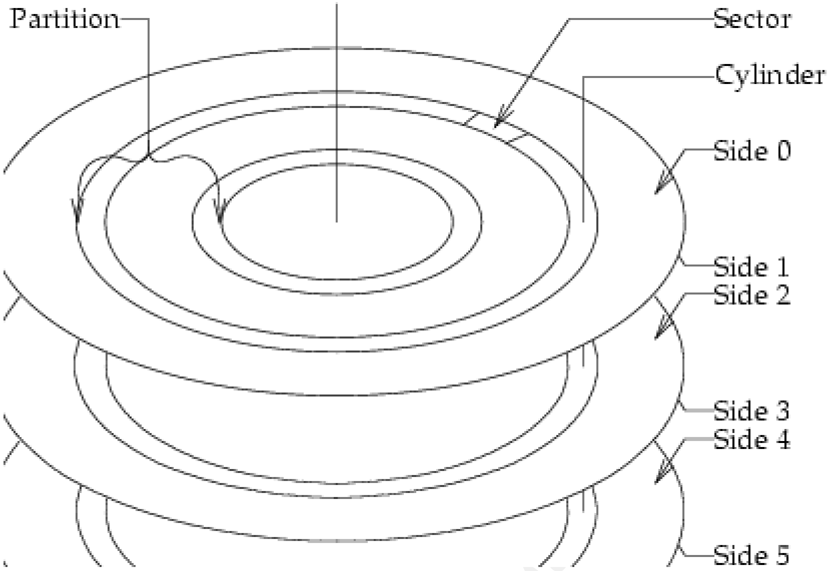
Figure 1 illustrates the surface of a disk platter as concentric rings called tracks, and the segmentation of tracks into sectors. For the purpose of discussion, we can assume that each sector can hold 512 bytes. A byte is 8 binary digits, or bits, which together represent the code for a specific alphanumeric or special character.

Figure 1 [11]



In order to make a physical hard disk useable, a partition table is created which defines one or more partition on the disk. Figure 2 helps illustrate that a partition is a collection of cylinders that the disk driving software and operating system present to the computer user as a disk drive such as C:\, D:\, and so on. If you wipe the contents of one partition you will not have wiped the partition table, master boot record, other partition. To address comprehensive disk wiping, the entire surface of the disk, not just a subset (a partition) that represents a portion of the disk.

Figure 2 [12]



Part of the nature of DASD architecture is that material[2] that is "deleted" is not actually destroyed, but is instead abandoned. Deleting a file simply flags those portions of the disk that store the file as available to store any other digital material. The actual content of the deleted file remains untouched until another file is written to that same location on the disk, overwriting the deleted material. This is like turning to a book's table of contents and erasing the line that lists chapter nine's title and page number. All that has been altered is the table of contents, no the pages of the actual chapter.

Since subsequent access to "deleted" material is trivial, ineffectively wiped [3] hard disks create a significant risk to material that the state is obligated to manage and protect. This includes question of exposure to litigation, and violation of HIPAA standards, along with potential damage to public's trust.

Today there are many commonly held misunderstandings about how to wipe a disk so that its contents are irrecoverable. For example;

- reformatting a disk destroys all the digital material is storing
- low-level formatting a disk will destroy its contents
- emptying and windows recycle bin, clearing logs, and deleting the contents of a file and re-saving it as an empty file will destroy a file.

do not impede later access to this same material.

In fact, reformatting a disk only alters its "table of contents" and not the actual material stored on the disk. While there are many references to "low-level formatting" on the web and in manuals, an actual low-level format is usually an option only to disk manufactures. Think of a hard disk as a bookshelf. You can rearrange the books as often as you want, and in any order you want. But this does not change the basic structure of the bookshelf. Reformatting rearranges the books, while a low-level format puts the shelves in place.

Faced with this problem, state agencies need a solution that;

1. effectively reduces the risk that digital material leaves the control of the owning agency to an acceptable level,
2. is inexpensive in purchase cost, required man hours, and use, and
3. is reasonable easy to perform.

To address the first requirement, the application of well tested techniques that make the retrieving of previous material impractical. Note that it is generally accepted that given enough money, time, expertise, and desire any deleted material can be retrieved. However, the goal is to reduce risk to an acceptable level rather than to eliminate the risk, which is usually impossible. The same principal can be seen in passwords. Any password can be cracked, given enough time and resources. The goal is to make the effort of cracking a password (or un-deleting a file) impractical due to the expense and time required. At that point, the few (people, nations…) who could consider making the huge investments necessary would use much less expensive techniques such as stealing the computer before you could delete your files.

To address the second requirement, this paper addresses the use of publicly available free software. While there are many products available that address this problem, they differ primarily in the range of options, reports, and techniques presented through a graphical interface.

Overview of the process

In developing this paper, several approaches were considered. While personally having practiced several methods of securely deleting digital material, most require an extensive understanding of hardware, software, and programming. This violates the third requirement.

So, attempts were made to automate several of these techniques. However, this created a catch-22 where there was then a need to be familiar with the components that "do" the automation.

Fortunately, a tool (Darik's Boot and Nuke, or "DBAN") was found that automates the best of these tools and techniques, satisfying the three requirements (effectiveness, low-cost, and ease of use).

After illustrating the use of DBAN, I will illustrate the use of some of these other individual tools and techniques. Of specific interest will be the use of dd to examine portions of a "wiped" disk to verify the effectiveness of the wipe process.

Details of the process

This paper recommends the use of DBAN, a freeware floppy diskette Linux distribution that automates the process of DASD discovery and the use of wipe to destructively overwrite the entire surface of hard disks.

Note that when finished, the wiped disk will no longer contain a partition table or operating system. In order to boot from, or use, the wiped disk, it must first be repartitioned and an operating system installed. The wipe process does not damage the disk in anyway, it simply removers *everything*! So while the slate has been completely wiped clean, it is still a perfectly good slate read for reuse.

Detailed walk-through of the process

The DBAN tool available at `http://`sourceforge.net/projects/dban/

As stated on the DBAN website;

```
Darik's Boot and Nuke ("DBAN") is a self-contained
boot floppy that securely wipes the hard disks of
most computers. DBAN will automatically and
completely delete the contents of any hard disk
that it can detect, which makes it an appropriate
utility for bulk or emergency data destruction.
```

1. **Download dban-0.4.0_i386.zip from http://sourceforge.net/projects/dban/ into an empty directory on your local system.**

2. **Unzip dban-0.4.0_i386.zip. This will produce the following files;**

```
C:\dbr\dban-0.4.0_i386>dir
 Volume in drive C has no label.
 Volume Serial Number is 3CBD-2ACD

 Directory of C:\dbr\dban-0.4.0_i386

02/27/2003  09:50 PM    <DIR>          .
02/27/2003  09:50 PM    <DIR>          ..
10/30/2002  04:48 PM             3,920 changelog.txt
02/27/2003  09:50 PM    <DIR>          config
10/30/2002  04:48 PM               103 Darik's Boot and Nuke.url
10/30/2002  04:48 PM         1,474,560 dban-0.4.0_i386.img
10/30/2002  04:48 PM             1,152 install.bat
02/27/2003  09:50 PM    <DIR>          licenses
10/30/2002  04:48 PM             1,016 notes.txt
10/30/2002  04:48 PM            17,863 rawrite2.exe
10/30/2002  04:48 PM             3,197 readme.txt
10/30/2002  04:48 PM             3,411 wipe.txt
               8 File(s)      1,505,222 bytes
```

3. Read the readme.txt file listed above. The readme.txt file contains clear instructions for creating a bootable floppy copy of the DBAN tool.

4. Place a blank floppy diskette in the floppy drive and run the install.bat script shown above. This batch file will place the DBAN system onto the floppy is a particular way. You cannot simply copy these file onto a floppy, they have to be placed on the floppy using the rawrite2.exe program also listed above.

5. Place your new DBAN diskette in the diskette drive of a system whose hard disk(s) you want to actually wipe.

6. Power on the machine to be wiped. Rather than loading from its hard disk, it will read the DBAN floppy and boot a small copy of Linux under which the wipe process will run.

7. Depending on the size and speed of the hard disk, the DBAN process will usually take several hours. The machine can be left unattended during this time.

8. When the process is finished, the machine can be powered off and released. Note that since the entire disk has been wiped clean, it no longer contains an operating system or partition table. So if you reboot from the wiped hard disk, the BIOS will complain in a variety of ways. To make the hard disk usable again, you will need to boot from a removable disk (floppy or CD) and repartition the disk, and then load an operating system.

Other tools and techniques

1. Download a copy of Trinux from http://trinux.sourceforge.net/ You will want the IDE version for running on a desktop, and the PCMCIA version for laptops. Also download the wipe.tgz package from http://trinux.sourceforge.net/pkg/
2. Following the instructions, build a bootable copy of the Trinux operating system.
3. After the Trinux floppy is built, copy the wipe.tgz file to the root of that floppy.
4. Boot with Trinux floppy. The boot process may take several minutes, especially when performing the DHCP initialization tasks.
5. When prompted for a "package diskette", type "Y" (without the quote marks) and allow the boot process to continue reading from the boot floppy. You should see the wipe.tgz package being loaded.
6. When prompted for "another package diskette", type "N" (without the quote marks) and let the boot process continue.
7. At the console prompt, enter the command "ls" and note any folder named "hd…" or "sd…". For example; hdb2, hda1, sda4, sdb2… These are the disks and partitions [8] that Trinux has recognized.
8. Enter the command "fdisk –l". This will identify the drives and partitions recognized.
9. Wipe (destructively overwrite) the entire surface of the physical disk. An example of running the dd command, **or** the wipe program;

```
for n in `seq 7`; do dd if=/dev/urandom of=/hda1; done
```

```
wipe -C 512 /dev/hdb2
```

10. If you want to recreate a partition on the wiped disk, there are many ways and tools available. For example; you can use the DOS fdisk program, an operating system install disk, or the following Linux command;

```
sfdisk --force --no-reread /dev/hdb1 < input.file

The input.file contents the following line;
            0,,83,*
```

11. With the disk repartitioned, you can then format the partition(s) into a file system such as DOS, NTFS, ext2, and ext3.

Verifying the results of the process

Once you have wiped a hard disk, you may want to verify for yourself that the material on the disk has been effectively overwritten.

Use dd to read a sector before wiping, and then the same after wiping. Compare the two. You can also use dd to "browse" the disk sector by sector inspecting the contents.

In the example below, dd will access disk hdb and pull one block of 512 bytes beginning at 9999 * 512 bytes from the beginning of the disk. This block will be dumped to the screen.

```
# dd if=/dev/hdb bs=512 count=1 skip=999
```

In this next example dd accesses the hda disk beginning 512 * 50 bytes from the beginning of the disk and will pull 512 * 2 bytes out and save them to the test.bin file for review.

```
# dd if=/dev/hda of=/test.bin bs=512 count=2 skip=50
```

References

1. There is a difference between *data* and *information*. Data is one or more recorded symbols or markings that is not presented in context. Information is data put into context. For example, the data "40601" is meaningless without a context. Then put into the context of "zip codes", that data meaningful.

2. *Material* is a general term describing the contents of digital storage. Examples of material include; a computer file, a specific section of a computer disk, any portion of a computer file.

3. The terms wiping, scrubbing, and sanitizing a hard disk referrers to the process of destructively overwriting the entire disk surface, replacing all material with different material. Effective wiping refers to writing repetitious patterns of specific binary values to the entire disk surface so as to make irrecoverable, in practical terms, the original material. See Peter Gutmann's paper Secure Deletion of Data from Magnetic and Solid-State Memory[7] for a detailed description of the patterns and their repetitions that are most effective for destructively overwriting digital material.

4. Trinux; "Trinux is a ramdisk-based Linux distribution that boots from a single floppy or CD-ROM, loads it packages from an HTTP/FTP server, a FAT/NTFS/ISO file system, or additional floppies."
   URL: http://trinux.sourceforge.net/

5. What You Don't See On Your Hard Drive Brian Kuepper, April 4, 2002. SANS Reading Room.
   URL: http://rr.sans.org/incident/don't_see.php

6. Secure File Deletion, Fact or Fiction? John R. Mallery. July 16, 2001. SANS Reading Room..
   URL: http://rr.sans.org/incident/deletion.php

7. Peter Gutmann; Secure Deletion of Data from Magnetic and Solid-State Memory
   URL: www.cs.auckland.ac.nz/~pgut001/secure_del.html

8. DBAN; "Darik's Boot and Nuke"
   URL: http://dban.sourceforge.net/

9. wipe - Secure Deletion Tool
   URL: http://wipe.sourceforge.net/

10. Linux Disks and Partitions
    URL: http://www.mandrakeuser.org/docs/mdoc/user/install-part-naming.html

11. image hd-schematic.gif
    URL: http://www.linuxvalley.it/encyclopedia/ldp/guide/sag/x754.html

12. image; img23.png
    URL: http://www.linuxman.com.cy/rute/node22.html