



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

Interested in learning more?

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

SANS GCFA Practical Assignment

Version 1.5

Author: Chris Russel
November, 2004

© SANS Institute 2005, Author retains full rights.

Contents

PART ONE – ANALYZE AN UNKNOWN IMAGE.....	4
SCENARIO INTRODUCTION.....	4
EXAMINATION DETAILS	4
<i>Timeline</i>	4
<i>Forensics Equipment and Tools</i>	5
<i>Methodology</i>	6
IMAGE DETAILS	6
<i>Verification of Image Integrity</i>	6
<i>Image Analysis</i>	7
<i>Metadata Analysis</i>	8
<i>Timeline Analysis</i>	9
<i>Recovering Deleted Files</i>	12
<i>Unallocated Space Analysis</i>	13
<i>File Content Analysis</i>	14
PROGRAM IDENTIFICATION AND FORENSIC DETAILS	15
<i>Unknown Binary File Analysis</i>	15
<i>Search for Code</i>	16
<i>Running Camouflage</i>	16
<i>Comparison of Binary Files</i>	18
<i>Usage of the Camouflage Program</i>	19
<i>Decryption and Examination of Policy File Contents</i>	20
SUMMARY AND CONCLUSION	23
LEGAL IMPLICATIONS	24
<i>Theft</i>	24
<i>Copyright</i>	24
<i>Mischief and Unauthorized Use</i>	24
ADDITIONAL INFORMATION	25

© SANS Institute 2005. Author retains full rights.

PART TWO – FORENSIC ANALYSIS OF A SYSTEM.....	26
INTRODUCTION.....	26
SYNOPSIS OF CASE FACTS	26
<i>Incident Background and Response Measures.....</i>	<i>26</i>
<i>System Placement Within the Network.....</i>	<i>27</i>
<i>Target System Description.....</i>	<i>29</i>
<i>Hardware Seized.....</i>	<i>29</i>
<i>Chain of Custody.....</i>	<i>30</i>
IMAGE MEDIA	30
<i>Imaging Procedure and Verification.....</i>	<i>30</i>
Step One – Attaching the Disk.....	31
Step Two – Checksum of Original Disk	31
Step Three – Copy Disk Image and Integrity Verification	31
Note on Swap Space.....	32
MEDIA ANALYSIS OF SYSTEM.....	32
Tools.....	32
<i>Comparison with Original Operating System Software.....</i>	<i>33</i>
Using Native Package Management Tools.....	33
<i>Examination of /etc</i>	<i>34</i>
<i>Start-up Files and Processes.....</i>	<i>34</i>
<i>Setuid/Setgid Files.....</i>	<i>37</i>
TIMELINE ANALYSIS.....	38
<i>System Installation and Updates.....</i>	<i>38</i>
<i>MAC Time Analysis.....</i>	<i>40</i>
RECOVER DELETED FILES	43
STRING SEARCH	46
<i>Keywords and Search Procedure.....</i>	<i>46</i>
CONCLUSIONS	47
<i>Synopsis of Intruder Activity</i>	<i>47</i>
<i>Method of Compromise.....</i>	<i>51</i>
<i>Summary.....</i>	<i>51</i>

PART ONE – ANALYZE AN UNKNOWN IMAGE

Scenario Introduction

As a computer investigation consultant, I have been hired by Ballard Industries to investigate evidence related to suspected industrial espionage. Recently, a competitor has begun manufacturing a fuel cell design that was once unique to Ballard, and was selling directly to many of Ballard's previous clients. It is suspected that the competitor, Rift Inc., somehow obtained copies of Ballard's customer database as well as proprietary engineering information related to the design and manufacturing of a certain fuel cell Ballard produces. An extensive investigation turned up only a single lead; a floppy disk was taken out of the Ballard R&D labs against company policy. I have been asked to examine the contents of the disk and provide a report.

Note: The floppy disk image was downloaded from the SANS website as filename: v1_5.gz, it has been renamed for the purposes of this scenario to l-260404-RJL1.img.gz. No other changes have been made.

Examination Details

Timeline

On the morning of 27 April 2004 I was contacted by the security administrator of Ballard, David Keen, for the purposes of acquiring my services for an examination of electronic evidence. At approximately 3pm on 27 April 2004 I signed a consultant agreement with Ballard which included the provision that I was to keep in confidence all details of the incident and any further information uncovered during my examination, with the resulting report to be provided to the security administrator.

After signing the agreement, at approximately 3:15pm, the known details of the incident were related to me:

During an investigation by Ballard into suspected industrial espionage, it was determined that Robert John Leszczynski, Jr. attempted to take a floppy disk out of the Ballard R&D labs at approximately 4:45pm on 26 April 2004. As this is against company policy, the security guard on duty seized the disk from Mr. Leszczynski's briefcase. According to procedure, the floppy disk was passed to the security administrator, David Keen.

The disk appears to contain documents regarding various Ballard policies, however due to its suspicious nature I have been asked to perform a detailed examination of the disk and establish if there is anything else on the disk and how it may have been used by Mr. Leszczynski.

The materials related to the incident were provided to me as follows:

- The floppy disk itself
- A CDROM (CD-R) containing the following files:
“fl-260404-RJL1.img.gz”

“fl-260404-RJL1.img.md5”

- Chain of custody form with the following information:

Evidence Tag #: fl-260404-RJL1
Description: 3.5 inch TDK floppy disk
Image filename: fl-260404-RJL1.img.gz
MD5 checksum:
d7641eb4da871d980adbe4d371eda2ad fl-260404-RJL1.img

David Keen also kept an identical copy of the CDROM with his notes.

After receiving the materials I returned to my lab to begin the examination.

Forensics Equipment and Tools

The following is a summary of the lab equipment and tools used in the investigation:

Forensics Workstation

Hardware: x86 architecture clone

Operating System: Debian GNU/Linux with 2.6 kernel

Software:

VMware Workstation 4.5 - provides a virtual machine environment.

Sleuthkit 1.71 - A development of The Coroner's Toolkit (a set of UNIX forensics tools by Wietse Venema and Dan Farmer) which has a number of additional capabilities, including the ability to understand FAT and NTFS filesystems. Sleuthkit tools are developed and maintained by Brian Carrier.

Autopsy 2.02 - A web-based interface to Sleuthkit tools which allows easier examination of some of the raw data.

VMware Windows Environment

VMware is important since it provides the virtual machine environment for some aspects of the examination. In particular it can be used to provide an isolated “sandbox” for testing the effects of unknown code on a running system. It also allows “snapshots” of the virtual machine environment for rapid testing and reversing the effects of any changes to the system, which would take much longer on a real machine.

Certain settings are noteworthy to prevent any unintended interaction outside the vmware sandbox:

- Network setting is Host-Only – on the Linux host, any packets being sent from the virtual host (regardless of operating system) will appear on the vmnet1 virtual adapter, but will go no further unless the Linux host is intentionally set up to do routing. This makes it simple to monitor any network activity or simulate network connections with the virtual machine without allowing packets onto a real network.
- Shared Folders are Disabled – this is a convenience setting which allows windows virtual hosts to easily share files with the real host. Undesirable in a forensics environment.

- Floppy Disk set to NOT connect at power on – since we may be using the virtual floppy, this is to prevent any accidental boot of the virtual machine off an unknown floppy image during the examination.

Methodology

The first few steps in the examination are outlined below:

- Make working copy of image and verify it is identical to original
- Record general filesystem type and directory information
- Detect deleted files and/or content in unallocated areas
- Review files
- String detection and analysis
- Binary detection and analysis

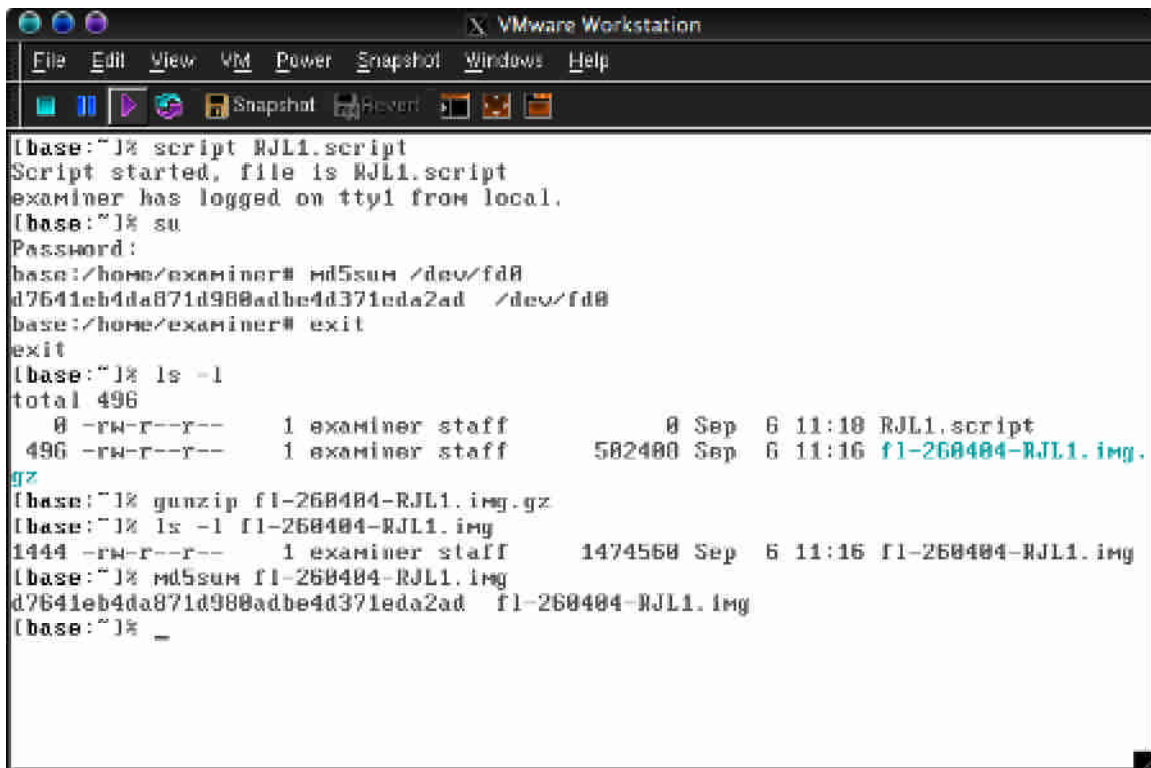
What is found in the first steps will determine what is done in the advanced stages of the examination.

Image Details

Verification of Image Integrity

To verify the floppy disk and image file are identical, md5sums are calculated for both

Note: for the purposes of this assignment, the image file is connected to the VMware floppy disk, and will show as /dev/fd0 in the linux virtual environment. This is to demonstrate that in a real investigation, the actual floppy disk would be used for direct comparison with the image file.



```
[base:~]% script RJL1.script
Script started, file is RJL1.script
examiner has logged on tty1 from local.
[base:~]% su
Password:
base:/home/examiner# md5sum /dev/fd0
d7641eb4da871d988adb4d371eda2ad /dev/fd0
base:/home/examiner# exit
exit
[base:~]% ls -l
total 496
  0 -rw-r--r--    1 examiner staff      0 Sep  6 11:10 RJL1.script
496 -rw-r--r--    1 examiner staff 502400 Sep  6 11:16 fl-260404-RJL1.img.gz
[base:~]% gunzip fl-260404-RJL1.img.gz
[base:~]% ls -l fl-260404-RJL1.img
1444 -rw-r--r--    1 examiner staff 1474560 Sep  6 11:16 fl-260404-RJL1.img
[base:~]% md5sum fl-260404-RJL1.img
d7641eb4da871d988adb4d371eda2ad fl-260404-RJL1.img
[base:~]% _
```

Figure 1 – MD5 Sum of Floppy Disk and Image File

As the screenshot shows, the MD5 sums match. The MD5 also agrees with the number David Keen entered on the chain of custody form, and can therefore be accepted to be an identical digital copy. The floppy disk is now filed in the locked evidence drawer. All further examination will use the image file instead to avoid damaging the original evidence.

Image Analysis

Without making any assumptions about the disk image, we'll start with the UNIX *file* command which should give us an accurate starting point.

```
% file fl-260404-RJL1.img
fl-260404-RJL1.img: x86 boot sector, code offset 0x3c, OEM-ID "mkdosfs", root
entries 224, sectors 2872 (volumes <=32 MB), sectors/FAT 9, serial number
0x408bed14, label: "RJL", FAT (12 bit)
```

It appears to be an MSDOS formatted floppy, using the FAT-12 filesystem. Now that we know the filesystem, further information can be obtained using the Sleuthkit *fsstat* command:

```
% fsstat -f fat12 fl-260404-RJL1.img
FILE SYSTEM INFORMATION
-----
File System Type: FAT

OEM Name: mkdosfs
Volume ID: 0x408bed14
Volume Label (Boot Sector): RJL
Volume Label (Root Directory): RJL
```

File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)

Total Range: 0 - 2871

* Reserved: 0 - 0

** Boot Sector: 0

* FAT 0: 1 - 9

* FAT 1: 10 - 18

* Data Area: 19 - 2871

** Root Directory: 19 - 32

** **Cluster Area: 33 - 2871**

META-DATA INFORMATION

Range: 2 - 45426

Root Directory: 2

CONTENT-DATA INFORMATION

Sector Size: 512

Cluster Size: 512

Total Cluster Range: 2 - 2840

FAT CONTENTS (in sectors)

105-187 (83) -> EOF

188-250 (63) -> EOF

251-316 (66) -> EOF

317-918 (602) -> EOF

919-1340 (422) -> EOF

1341-1384 (44) -> EOF

This listing confirms the filesystem type, and shows the volume label “RJL” which are the initials of the individual the disk belonged to. It also shows an overview of the content in the FAT (File Allocation Table); it appears there are 6 files with allocated blocks. Potentially noteworthy: All of the allocated blocks are contiguous, but the first one starts at sector 105. Since the data area of the disk (not including the root directory) starts at sector 33, there is very likely a deleted file(s) in the unallocated sectors 33-104.

Metadata Analysis

To get a quick, readable, overview of what is on the disk without actually mounting it, we use the Sleuthkit *fls* command to list all directory entries, which will include entries left over from deleted files.

```
% fls -f fat12 -r fl-260404-RJL1.img
r/r 3:  RJL                (Volume Label Entry)
r/r * 5:  CamShell.dll    (_AMSHLL.DLL)
r/r 9:  Information_Sensitivity_Policy.doc (INFORM~1.DOC)
r/r 13: Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
r/r 17: Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
r/r 20: Password_Policy.doc (PASSWO~1.DOC)
r/r 23: Remote_Access_Policy.doc (REMOTE~1.DOC)
r/r 27: Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
```

```
r/r * 28:      _ndex.htm
```

In the FAT filesystem, when files are deleted, typically the first character of the directory entry is replaced with an underscore. There are 2 directory entries (_AMSHLL.DLL and _ndex.htm) that refer to deleted files and 6 files still allocated.

To verify the directory information, we can mount the disk image under Linux in read-only mode and list the files using regular system commands. The mount command uses “ro” to ensure nothing is accidentally modified within the disk image. The “noatime” tells the system not to update access timestamps on any of the files – this is redundant when used with the ro flag but is a habit. VFAT is the Linux nomenclature for the family of FAT filesystems. The disk image contents will be available in /mnt.

```
# mount -o ro,noatime,loop -t vfat fl-260404-RJL1.img /mnt
# ls -l /mnt
total 640
 22 -rwxr--r--  1 root root  22528 2004-04-23 14:10
Acceptable_Encryption_Policy.doc*
 42 -rwxr--r--  1 root root  42496 2004-04-23 14:11
Information_Sensitivity_Policy.doc*
 32 -rwxr--r--  1 root root  32256 2004-04-22 16:31
Internal_Lab_Security_Policy1.doc*
 33 -rwxr--r--  1 root root  33423 2004-04-22 16:31
Internal_Lab_Security_Policy.doc*
301 -rwxr--r--  1 root root 307935 2004-04-23 11:55
Password_Policy.doc*
211 -rwxr--r--  1 root root 215895 2004-04-23 11:54
Remote_Access_Policy.doc*
```

Note that the user and group ownership and permission bits are substituted by the Linux filesystem driver; the FAT filesystem has no concept of ownership and very limited permission flags which cannot be translated to UNIX permissions. Ownership will be the same default settings for all files in the mount.

We calculate the MD5s of the files and copy them off the image for future reference, if needed. We do not need it mounted for any further analysis.

```
# md5sum /mnt/*
f785ba1d99888e68f45dabeddb0b4541  /mnt/Acceptable_Encryption_Policy.doc
99c5dec518b142bd945e8d7d2fad2004
/mnt/Information_Sensitivity_Policy.doc
e0c43ef38884662f5f27d93098e1c607
/mnt/Internal_Lab_Security_Policy1.doc
b9387272b11aea86b60a487fbdclb336  /mnt/Internal_Lab_Security_Policy.doc
ac34c6177ebdc4f4adc41f0e181be1bc  /mnt/Password_Policy.doc
5b38d1ac1f94285db2d2246d28fd07e8  /mnt/Remote_Access_Policy.doc
# cp /mnt/* .
# umount /mnt
```

Timeline Analysis

The FAT filesystem does have the capability to record the usual access timestamps (last written/modified, last access, creation), however there are some caveats, as mentioned in the Sleuthkit FAT implementation notes:

- Last written times are required and are accurate to the second

- Last access time is optional and is only accurate to the day (time will always be 00:00:00)
- Creation time is optional but is accurate to the tenth of a second

To gather the timeline information, the Sleuthkit *fls* command is used, this time with the *-m* option to print the output with the MAC timestamps. According to standard procedure, we also add the output of the *ils* command, which gathers more information on unallocated inodes (presumably belonging to deleted files). Due to the simplistic way FAT deals with metadata there are no inodes, only directory entries indexed by starting cluster. Since *fls* can already detect deleted files from the directory entries, we expect *ils* to show the same information for deleted files.

© SANS Institute 2005, Author retains full rights

```

[icarus:~/evidence]# fls -f fat12 -m / -r fl-260404-RJL1.img > body
[icarus:~/evidence]# ils -f fat12 -m fl-260404-RJL1.img >> body
[icarus:~/evidence]# mactime -b body 01/01/1970
Sat Feb 03 2001 19:44:16 36864 m.. -rwxrwxrwx 0 0 5 <fl-260404-RJL1.img- AMSHELL.DLL-dead-5>
36864 m.. -/-rwxrwxrwx 0 0 5 /CamShell.dll (_AMSHLL.DLL) (deleted)
Thu Apr 22 2004 16:31:06 32256 m.. -/-rwxrwxrwx 0 0 13 /Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
33423 m.. -/-rwxrwxrwx 0 0 17 /Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
Fri Apr 23 2004 10:53:56 727 m.. -/-rwxrwxrwx 0 0 28 /_ndex.htm (deleted)
727 m.. -rwxrwxrwx 0 0 28 <fl-260404-RJL1.img- _ndex.htm-dead-28>
Fri Apr 23 2004 11:54:32 215895 m.. -/-rwxrwxrwx 0 0 23 /Remote_Access_Policy.doc (REMOTE~1.DOC)
Fri Apr 23 2004 11:55:26 307935 m.. -/-rwxrwxrwx 0 0 20 /Password_Policy.doc (PASSWO~1.DOC)
Fri Apr 23 2004 14:10:50 22528 m.. -/-rwxrwxrwx 0 0 27 /Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
Fri Apr 23 2004 14:11:10 42496 m.. -/-rwxrwxrwx 0 0 9 /Information_Sensitivity_Policy.doc (INFORM~1.DOC)
Sun Apr 25 2004 00:00:00 0 .a. -/-rwxrwxrwx 0 0 3 /RJL (Volume Label Entry)
Sun Apr 25 2004 10:53:40 0 m.c -/-rwxrwxrwx 0 0 3 /RJL (Volume Label Entry)
Mon Apr 26 2004 00:00:00 215895 .a. -/-rwxrwxrwx 0 0 23 /Remote_Access_Policy.doc (REMOTE~1.DOC)
33423 .a. -/-rwxrwxrwx 0 0 17 /Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
36864 .a. -rwxrwxrwx 0 0 5 <fl-260404-RJL1.img- AMSHELL.DLL-dead-5>
307935 .a. -/-rwxrwxrwx 0 0 20 /Password_Policy.doc (PASSWO~1.DOC)
22528 .a. -/-rwxrwxrwx 0 0 27 /Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
42496 .a. -/-rwxrwxrwx 0 0 9 /Information_Sensitivity_Policy.doc (INFORM~1.DOC)
727 .a. -/-rwxrwxrwx 0 0 28 /_ndex.htm (deleted)
36864 .a. -/-rwxrwxrwx 0 0 5 /CamShell.dll (_AMSHLL.DLL) (deleted)
32256 .a. -/-rwxrwxrwx 0 0 13 /Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
727 .a. -rwxrwxrwx 0 0 28 <fl-260404-RJL1.img- _ndex.htm-dead-28>
Mon Apr 26 2004 09:46:18 36864 .c -rwxrwxrwx 0 0 5 <fl-260404-RJL1.img- AMSHELL.DLL-dead-5>
36864 .c -/-rwxrwxrwx 0 0 5 /CamShell.dll (_AMSHLL.DLL) (deleted)
Mon Apr 26 2004 09:46:20 42496 .c -/-rwxrwxrwx 0 0 9 /Information_Sensitivity_Policy.doc (INFORM~1.DOC)
Mon Apr 26 2004 09:46:22 32256 .c -/-rwxrwxrwx 0 0 13 /Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
Mon Apr 26 2004 09:46:24 33423 .c -/-rwxrwxrwx 0 0 17 /Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
Mon Apr 26 2004 09:46:26 307935 .c -/-rwxrwxrwx 0 0 20 /Password_Policy.doc (PASSWO~1.DOC)
Mon Apr 26 2004 09:46:36 215895 .c -/-rwxrwxrwx 0 0 23 /Remote_Access_Policy.doc (REMOTE~1.DOC)
Mon Apr 26 2004 09:46:44 22528 .c -/-rwxrwxrwx 0 0 27 /Acceptable_Encryption_Policy.doc (ACCEPT~1.DOC)
Mon Apr 26 2004 09:47:36 727 .c -/-rwxrwxrwx 0 0 28 /_ndex.htm (deleted)
727 .c -rwxrwxrwx 0 0 28 <fl-260404-RJL1.img- _ndex.htm-dead-28>

```

Another limitation of FAT is that the timestamps are simply written to disk as string with date, hour, minute, etc., not using a universal epoch starting point such as UNIX. This means that we need to know the timezone of the system the disk was used in to verify the times, even if the clock on the computer is set correctly. Of course, in this case we do not know what system it was used in (although it seems highly likely to be a computer in Ballard R&D). Ballard security will need to check what the time settings on those systems are (including to check the timezone is set correctly) to determine if these timestamps can be considered accurate.

Assuming that the timestamps are accurate, it can be seen that all the files were copied to the floppy disk on the morning of 26 April 2004, starting at 9:46:18am. All the document files and the camshell.dll file appear to have been copied over together, at the timestamps are only seconds apart. The number of seconds between them roughly correlates to the size of the files and the time it takes for a 3.5 inch floppy disk to write data. The index.htm file is nearly a minute later which implies there was some delay (of less than a minute) in the copying of that file compared to all the others.

The last write time of the volume label indicates when the disk was formatted, the previous day, Sunday 25 April 2004 10:53:40am. Ballard security will have to check if Mr. Leszczynski was at work on Sunday, otherwise it is likely the disk was formatted on his home computer or other non-company system.

The last write time of the rest of the files indicates all the files except camshell.dll had been worked on the previous Thursday and Friday.

Since the create times are later than the modification times, these files were probably last written to on a computer's hard disk or other media (Thursday and Friday), and then copied to the floppy on the monday. The exception is the camshell.dll file which was created in 2001.

Recovering Deleted Files

We also need to get copies of the deleted files (CAMSHELL.DLL and index.htm). *icat* can be used to extract them, provided no other files have written over the clusters that were allocated to the files originally. *Istat* is used to determine where on disk the files are located and if there is any overlap with other deleted files.

```
# istat -f fat12 fl-260404-RJL1.img 5
Directory Entry: 5
Not Allocated
File Attributes: File, Archive
Size: 36864
Num of links: 0
Name: _AMSHHELL.DLL

Directory Entry Times:
Written:      Sat Feb  3 19:44:16 2001
Accessed:     Mon Apr 26 00:00:00 2004
Created:      Mon Apr 26 09:46:18 2004

Sectors:
33
```

Recovery:

```
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104
# istat -f fat12 fl-260404-RJL1.img 28
```

Directory Entry: 28

Not Allocated

File Attributes: File, Archive

Size: 727

Num of links: 0

Name: _ndex.htm

Directory Entry Times:

Written: Fri Apr 23 10:53:56 2004

Accessed: Mon Apr 26 00:00:00 2004

Created: Mon Apr 26 09:47:36 2004

Sectors:

33

Recovery:

33 34

Note the highlighted “recovery” sectors. The index.htm file is written over the first 2 sectors of the CamShell.dll file and then deleted, about 1 minute after CamShell.dll is copied to the floppy disk and deleted. Therefore, the start of the CamShell.dll file is not recoverable, but the rest of it is.

Icat is used to recover the files to the extent possible:

```
icat -f fat12 -r fl-260404-RJL1.img 5 > CamShell.dll
```

```
icat -f fat12 -r fl-260404-RJL1.img 28 > index.htm
```

File sizes of the resultant files agree with the FAT directory entries:

```
36864 CamShell.dll
```

```
727 index.htm
```

Again, the first 727 bytes of the resultant CamShell.dll will be the same as the index.htm file.

Unallocated Space Analysis

To ensure nothing is missed on the disk surface we look at unallocated sectors.

According to the disk FAT (from earlier fsstat), sectors 1385 to the end of the disk are not allocated by any directory entry, including deleted ones. This surface can be examined with dd or dls to see if there is any interesting content – dls is the easier of the two for this type of search since it will automatically skip allocated blocks:

```
# dls -f fat12 fl-260404-RJL1.img 1385- | sum
00000 744
```

The UNIX *sum* command shows the sum of all the bytes read by the program and the number of 1k blocks read. It clearly shows the entire remainder of the disk (744k) is zero-bytes.

Another command will search the slack space. With only 6 allocated files and 512 bytes blocks, there cannot be more than 3k bytes in total. It can be extracted with the *dls* command *-s* option:

```
# dls -s -f fat12 fl-260404-RJL1.img | sum
00000      2
```

Neither the unallocated or the slack space are of any interest. It is possible there is something in the slack space of the two deleted files, in fact in the slack space of the *index.htm* is some of the *CamShell.dll* file as we have already determined. At the end of the *CamShell.dll* file there is $512 - (36864 \bmod 512) = 18$ bytes of slack space.

dd and *tail* are an easy way to extract this data. The skip argument is used to specific which sector we want.*dd*

```
# dd if=fl-260404-RJL1.img bs=512 skip=104 count=1 | tail -c 18 |
hexcat
1+0 records in
1+0 records out
512 bytes transferred in 0.000102 seconds (5020383 bytes/sec)
00000000 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
00000010 - 00 00 ..
```

The data is all zero. It is clear that the unallocated space and slack space were not used to hide any data. We now focus on the highly suspicious *camshell.dll* file and document files.

File Content Analysis

One curiosity is the existence of the two files with the same/similar name:

```
Thu Apr 22 2004 16:31:06      32256 m.. -/-rwxrwxrwx 0          0          13
/Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
                        33423 m.. -/-rwxrwxrwx 0          0          17
/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
```

These files are also the first of the document files to be last written. They have the same last written timestamp (to the second), which suggests that some automated process was used to create them. The differences between the two files may be instructive in the examination of the others.

Using the *strings* command on both of them, it appears that the readable text is simply the Ballard policy. A copy of the actual policy would be useful to compare with to see if either file is original. Using the UNIX *head* command I compare the first 32256 bytes of the larger file to the entire smaller one:

```
[icarus:~/evidence]% cat Internal_Lab_Security_Policy1.doc | md5sum
e0c43ef38884662f5f27d93098e1c607 -
[icarus:~/evidence]% head -c 32256 Internal_Lab_Security_Policy1.doc |
md5sum
e0c43ef38884662f5f27d93098e1c607 -
```

The MD5s match, indicating the content is identical except for the last 33423 – 32256 = 1167 bytes. We use the UNIX command

```
tail -c 1167 Internal_Lab_Security_Policy.doc | strings
```

It appears to be unreadable binary. Viewing with the *hexcat* command

```
tail -c 1167 Internal_Lab_Security_Policy.doc | hexcat
00000000 - 20 00 b9 28 c4 01 60 38 b8 73 75 29 c4 01 e0 36
.(f.`8•su)f..6
00000010 - 98 ba b9 28 c4 01 30 ff 7b 7f 38 01 00 00 4b b5
..•(f.0.{.8...Kµ
00000020 - 1b 4f 2c d1 7d 8d 8d a6 d1 02 00 1b f1 93 e9 17
.O,-}...-.....
00000030 - 25 25 92 92 10 74 d4 4c b8 5e eb 40 93 94 c2 94
%%...t`L•^.@...¬.
00000040 - 22 a0 27 6e 6f ae 9e f5 6e 0f 59 00 4d 6f 3b ff
".'noÆ..n.Y.Mo;.
(continues...)
```

shows that approximately the first third of the extra bytes are indeed binary and do not have any obvious pattern or meaning. The final two-thirds of the bytes are almost entirely character 0x20, or the ASCII “space” character. The first part could be encrypted content, but the non-random remaining part appears mostly content-free.

Next step is to look at camshell.dll.

Program Identification and Forensic Details

Unknown Binary File Analysis

First, running CamShell.dll through strings -10 shows some familiar identifiers:

```
... (deleted)
GC:\WINDOWS\SYSTEM\MSVBVM60.DLL\3
FIShellExtInit
C:\My Documents\VB Programs\Camouflage\Shell\IctxMenu.tlb
... (more)
__vbaStrVarCopy
__vbaAryUnlock
__vbaFreeStr
__vbaFreeObj
CamShell.dll
DllCanUnloadNow
DllGetClassObject
DllRegisterServer
DllUnregisterServer
stdole2.tlbWWW
IctxMenu.tlbWWW
1CamouflageShellW
_ShellExtWWWd
... (etc)
```

“vba” strings are Microsoft Visual Basic functions which are part of the dll, which is a clue how this file was made, not to mention the full path indicated above.

“CamouflageShell” seems to indicate the real name of the program as it is similar to the

filename. "ShellExtWWW" may imply that this program somehow hooks into web browsing functionality.

Search for Code

A google search for "Camouflage Shell" does not turn up any computing-related items. "camshell.dll" turned up only a single link. "camouflage dll" turned up several, including this site:

<http://camouflage.unfiction.com/>

The overview includes a useful description:

What is Camouflage?

Camouflage allows you to hide files by scrambling them and then attaching them to the file of your choice. This camouflaged file then looks and behaves like a normal file, and can be stored, used or emailed without attracting attention.

For example, you could create a picture file that looks and behaves exactly like any other picture file but contains hidden encrypted files, or you could hide a file inside a Word document that would not attract attention if discovered. Such files can later be safely extracted.

For additional security you can password your camouflaged file. This password will be required when extracting the files within. You can even camouflage files within camouflaged files.

Camouflage was written for use with Windows 95, Windows 98, Windows ME, Windows NT and Windows 2000, and is simple to install and use.

There is no source code on the site, but there is an EXE installer for download. After downloading this, we copy it along with the CamShell.dll from the floppy image to our VMware environment for further study.

Running Camouflage

Our objective is to install Camouflage while monitoring what changes are made to the system. First we save the VMware snapshot, then run Winalysis and take a snapshot within that program.

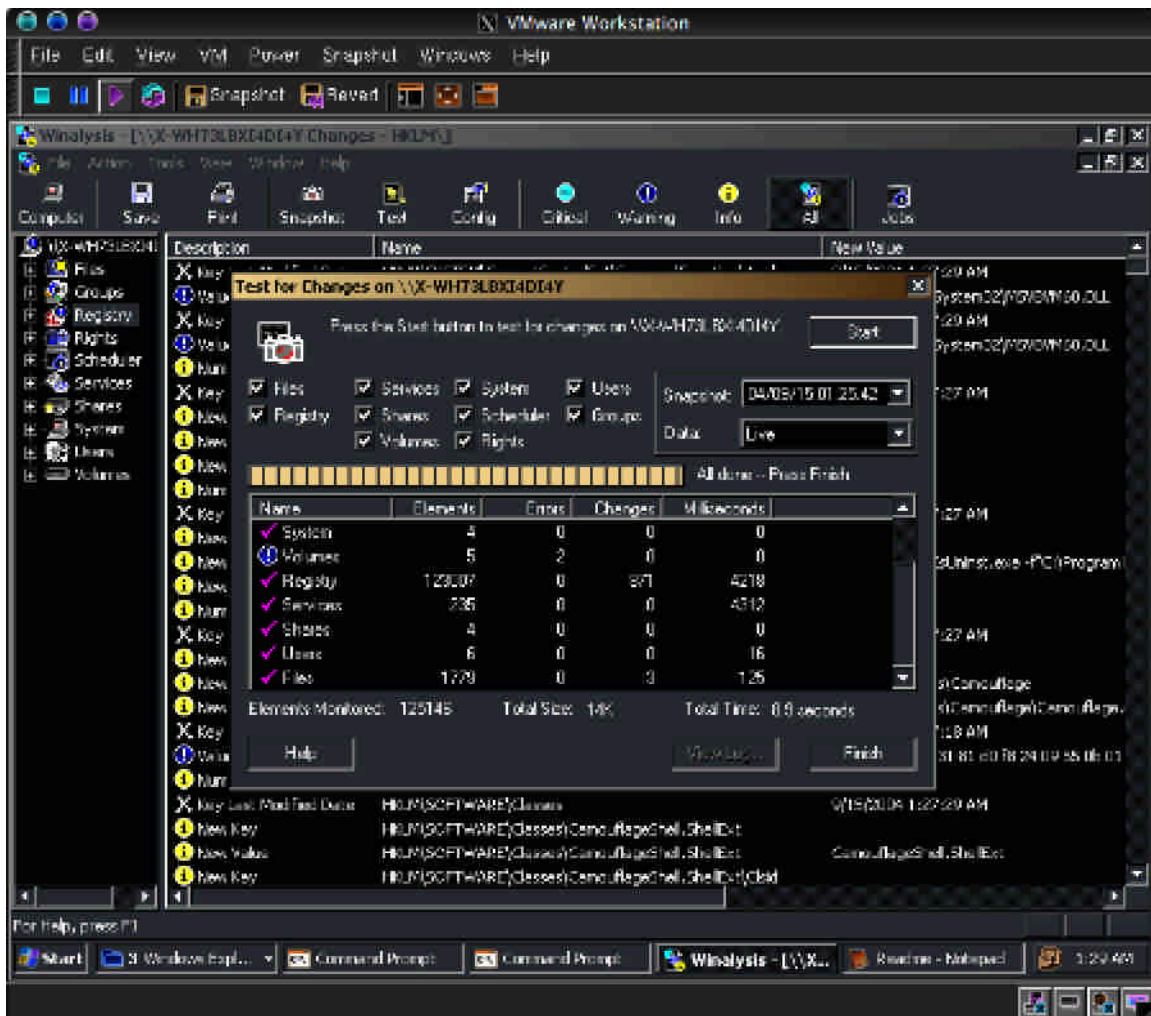


Figure 2 – Winanalysis snapshot of changes after Camouflage install

Although there are 871 registry changes, there are only 3 file changes in the windows directories. Investigating those shows a new file:

C:\WINDOWS\System32\MSCOMCTL.OCX

Also Camouflage has installed itself in the “Program Files” directory. Viewing this shows the CamShell.dll file which has a file size identical to the CamShell.dll found on the floppy disk.



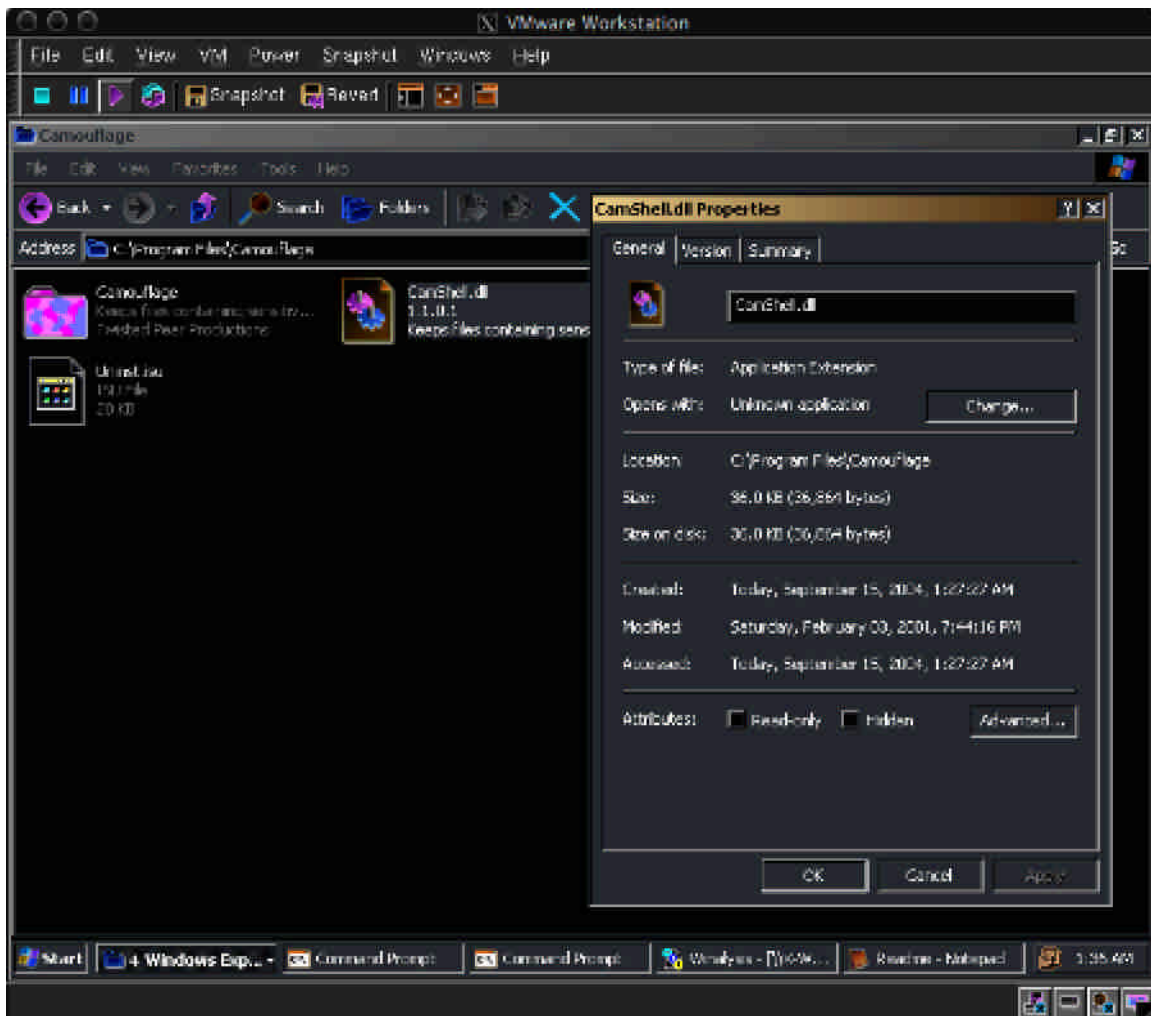


Figure 3 – CamShell.dll after Camouflage Installation

Comparison of Binary Files

The new CamShell.dll file is transferred back to the Linux environment for comparison with the version on the floppy disk. MD5 sums show they do not match (as expected since the index.htm file overwrote the first 727 bytes of the copy on the floppy disk). However, if the first 727 bytes are ignored for the comparison ($36864 - 727 = 36137$ bytes), the MD5 sums match.

© SANS Institute

```

X| russel@icarus [~/unknown]
[icarus:~/unknown]% ls -l CamShell*
36 -rw-r----- 1 russel users 36864 2004-09-15 01:44 CamShell.dll
36 -rw-r--r-- 1 russel users 36864 2004-09-14 23:01 CamShell.dll-floppy
[icarus:~/unknown]% md5sum CamShell*
4e986ab0909d2946bed868b5f896906f CamShell.dll
6462fb3acca0301e52fc4ffa4ea5eff8 CamShell.dll-floppy
[icarus:~/unknown]% tail -c 36137 CamShell.dll | md5sum
ab16749d6fb4cc35b004319e7f4abb50 -
[icarus:~/unknown]% tail -c 36137 CamShell.dll-floppy | md5sum
ab16749d6fb4cc35b004319e7f4abb50 -
[icarus:~/unknown]% █

```

Figure 4 – CamShell.dll MD5s

If these files were not the same binary it would be nearly impossible for the MD5s to match perfectly on the last 36137 bytes, not to mention having the same filename and file size. There is no doubt CamShell.dll on the floppy disk is a component of the same Camouflage program as I downloaded from the Internet.

Usage of the Camouflage Program

The Camouflage program was written specifically to hide data inside benign-looking files, including JPEG image files and Microsoft Word documents. This would perfectly serve Mr. Leszczynski's interest in smuggling proprietary Ballard information out of the R&D labs.

Camouflage is a steganographic tool, which is capable of hiding any arbitrary data file into other common file types, including JPG images and Microsoft Word documents.

As a steganographic tool, Camouflage is not sophisticated. It works by simply appending an encrypted form of the additional data to the end of the file. This does not cause any problem with those file formats since they have internal structure delimiters that will prevent an image viewer or word processor program from trying to display the hidden data. This is a rather weak method of steganography since it is simple to detect the presence of this type of file addition if you know what to look for, for example by checking for data after the end-of-image delimiter in a JPEG file. Other forms of steganography such as modifying the least significant bits in an image are far more difficult to detect.

Further research using Google shows some evidence of a weakness in the way Camouflage hides data. A search for "Camouflage steganography" immediately shows a link of interest:

<http://www.guillermi2.net/stegano/camouflage/>

This web page, by “Guillermi”, describes an analysis of Camouflage’s data hiding technique in detail; including the weak method used by Camouflage to embed the password and hidden data, and how to extract that data. From the description, the “encryption” used by Camouflage is simple XOR of the data and a fixed key which is part of the program. It is out of the scope of this paper to include a large discussion of cryptography – interested readers can refer to Bruce Schneier’s “Applied Cryptography”. In short, XOR with a never-reused key is considered a “one-time pad” form of encryption which is difficult to break. Reusing the same key for every document makes this a “many-time pad” which can be easily cracked via a known-plaintext attack. However, even more simple than determining the entire encryption key, we can just decrypt the password since it is stored along with the document and encrypted the same way. After that, we can let Camouflage decrypt the rest itself.

Decryption and Examination of Policy File Contents

The password is stored at a fixed offset from the end of the file, at -275 bytes. Checking the Internal_Lab_Security_Policy.doc shows it is all 0x20 characters (ASCII space character), which means no password is set. As shown in Figure 5, using the “uncamouflage” option reveals hidden files in the document.

© SANS Institute 2005, Author retains full rights

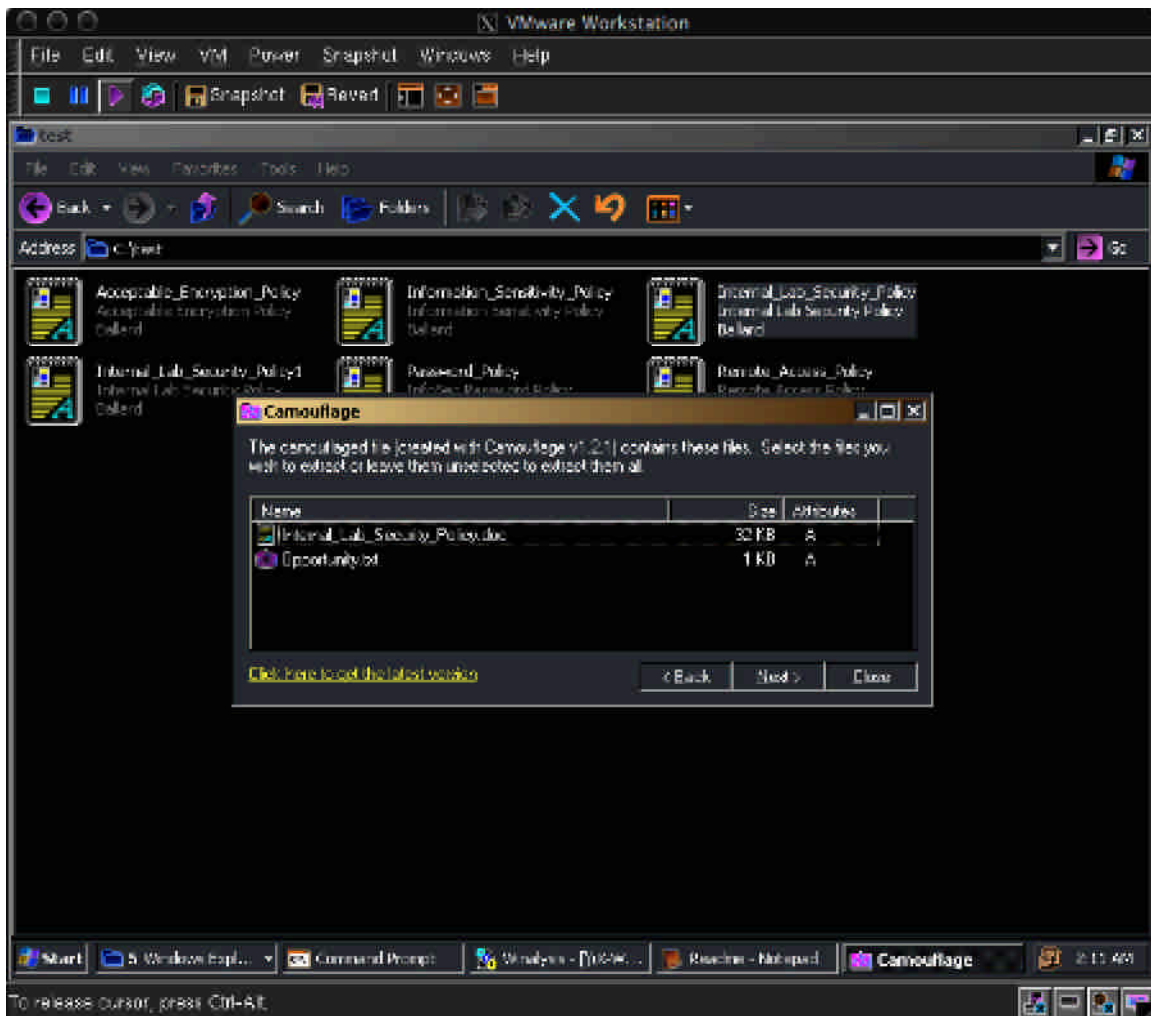


Figure 5 – “Uncamouflage” Success

The file “opportunity.txt” contains the text:

I am willing to provide you with more information for a price. I have included a sample of our Client Authorized Table database. I have also provided you with our latest schematics not yet available. They are available as we discussed - "First Name".
My price is 5 million.

Robert J. Leszczynski

The other files appear to be password protected. Again, looking at the location of the password in the other files (-275 byte offset from end) reveals the encrypted form of the password:

50f0 174d 78c3 2020 2020 2020

It is apparent the first 6 characters are the encrypted password, and all that is left is to XOR that value with the hard-coded string the camouflage program uses, as documented in the article mentioned above:

0295 7A22 0CA6 14E1

The operations to find the password are shown below:

Filename	Camouflage password string
Remote_Access_Policy.doc	50f0 174d 78c3 2020 2020 2020 XOR 0295 7A22 0CA6 14E1 = 5265 6d6f 7465 R e m o t e
Acceptable_Encryption_Policy.doc	None, no data hidden
Password_Policy.doc	52f4 0951 7bc9 6685 2020 2020 XOR 0295 7A22 0CA6 14E1 = 5061 7373 776f 7264 P a s s w o r d
Information_Sensitivity_Policy.doc	None, no data hidden
Internal_Lab_Security_Policy.doc	None, no data hidden

In these files there is an Access MDB file with customer data, and 3 image files including diagrams and a scanned page of notes regarding fuel cell design.

© SANS Institute 2005, Author retains full rights.

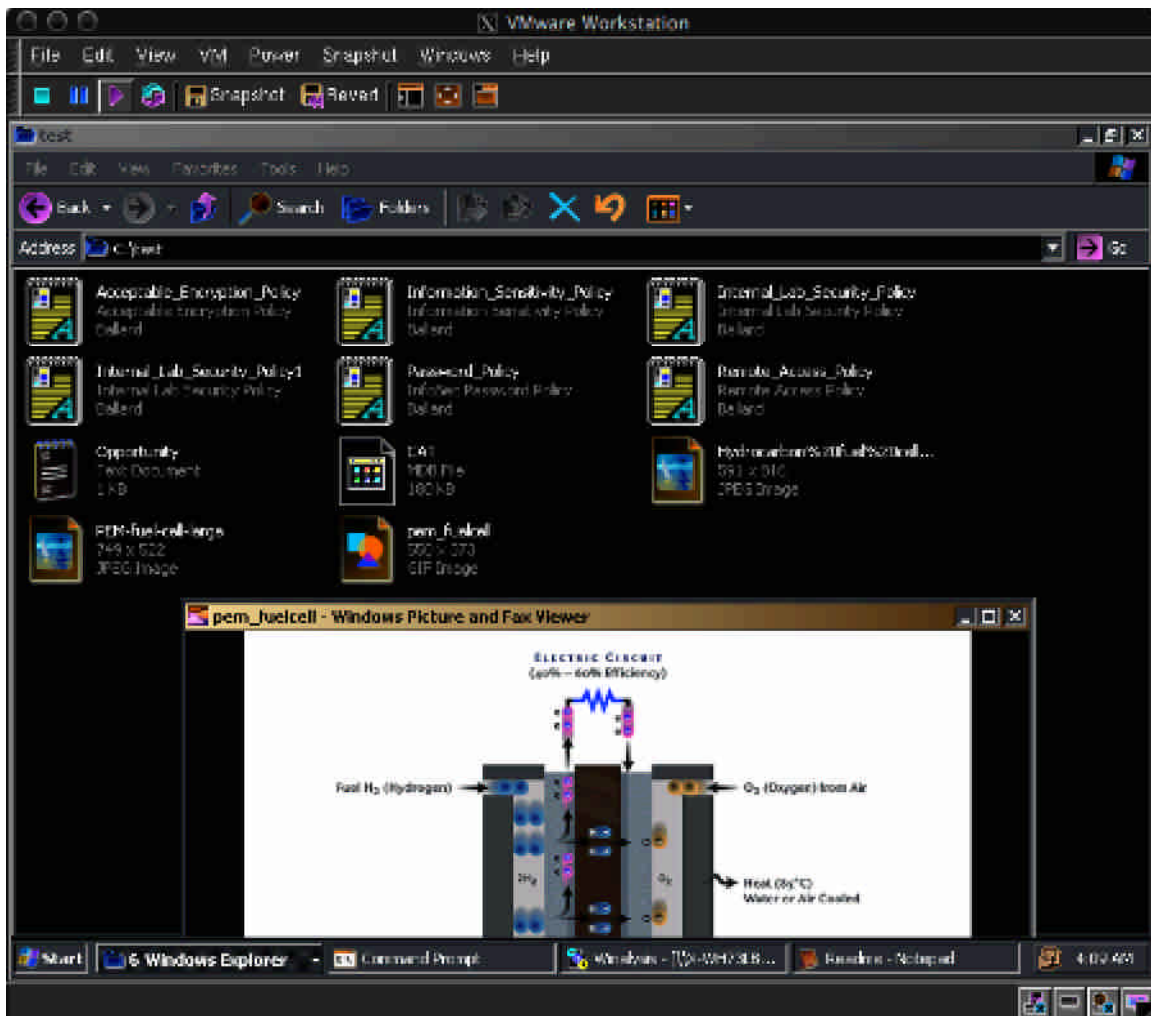


Figure 6 – Hidden documents

Summary and Conclusion

It appears Mr. Leszczynski is the data leak. From the Opportunity.txt file, he admits having provided information in the past, and a willingness to provide more such as has been recovered from the floppy disk. The information includes customer data and fuel cell design information taken from the Ballard R&D lab. If the security department had not noticed the policy violation of taking the floppy disk out of the lab and had not investigated it properly, this data theft would have gone unnoticed.

Although the security department showed vigilance, the system administrators of the R&D systems could have helped prevent this by locking down the configuration of all computers in the lab. Steps that could have mitigated this, or at least provide a chance of detection include:

- Disallow administrator logins to the R&D computers. This would prevent installation of any software which hooks into the system such as Camouflage does.

- Configure software restriction policies on the systems to prevent any unauthorized programs from being run on the system. This can be done by using path rules to allow windows and pre-installed software to run as expected, but deny any user-writable access to those paths, and restrict any other path from running code.
- Install host-based intrusion detection software such as Tripwire that can detect modifications to the system and automatically report suspicious activity to security.

Meanwhile all systems with confidential/proprietary data on them should be checked for the program used by Mr. Leszczynski. Camouflage is easy to detect by looking for the camshell.dll file anywhere in the system, and by default it is installed in “Program Files”.

Legal Implications

Discussion of the legal implications of this case will pertain to the jurisdiction of Ontario, Canada.

Theft

Although the use of steganographic software alone is not considered a crime, the actions allegedly taken by Mr. Leszczynski to use the Steganographic software for the purposes of stealing confidential information would be considered theft under the Criminal Code of Canada.

Section 322 of the Criminal Code deals with theft, and defines two possible charges: Theft over \$5000 and theft under \$5000. The only difference is the value of the property involved and the potential sentences. Theft of under \$5000 can result in a maximum sentence of 2 years in prison. Theft over \$5000 can result in a maximum sentence of 10 years in prison (Section 334). In this case, the company would certainly qualify this data theft under the latter charge.

Since the Criminal Code is written in a general fashion to apply widely to various issues, there is no written distinction between data which is property and any other type of property theft.

Copyright

Intellectual Property may also be covered under the Copyright Act, although the penalty would not be as great.

Stealing proprietary data and design blueprints, computer code, or other corporate information, potentially including the customer database (if the company has been able to copyright the *presentation* of that data, not the customer information) could be considered a copyright violation. This is a still-emerging area of law, particularly when dealing with computer data and databases. Since the idea of theft still carries with it the concept of the owner being deprived of something, copying a database or information file may not always be considered theft provided the owner has not actually lost anything from their copy. In this case, the theft could also be interpreted as a violation of the Copyright Act (Section 42). This act includes penalties of up a \$1,000,000 fine and 5 years in prison for copyright violations.

Mischief and Unauthorized Use

Finally, an additional consideration which may also apply, particularly if there is no evidence of data theft, but only misuse of the computer, is Section 342.1 – Unauthorized Use of Computer. This is a recent addition to the Criminal Code which is mainly intended to address computer viruses and computer intrusions. As such, it applies when the computer misuse is for the purposes of Mischief (Section 430). Mischief covers any unauthorized attempt or success in modifying or destroying data, and/or adversely affecting the operation of computer systems. The maximum sentence is 10 years in prison.

Additional Information

For further information on the topics in this paper, the following sources are recommended:

Forensics Tools	VMware Virtual Machine Software, http://www.vmware.com/ Sleuthkit & Autopsy Forensics Analysis, http://www.sleuthkit.org/ Winalysis, http://www.winalysis.com/
Cryptography	Schneier, Bruce. “Applied Cryptography, 2 nd ed.” Wiley & Sons, Inc. 1996.
Steganography	Cole, Eric. “Hiding in Plain Sight”. Wiley & Sons, Inc. 2003. Provos, Niels. “Defeating Statistical Steganalysis”. http://niels.xtdnet.nl/stego/ Provos, Niels. “OutGuess – Universal Steganography”. http://www.outguess.org/
Host-based Intrusion Detection	Tripwire. http://www.tripwiresecurity.com/
Camouflage	Guillermi. “Easily Breaking a Very Weak Steganography Software: Camouflage”. http://www.guillermi2.net/stegano/camouflage/index.html
Legal Implications	Criminal Code of Canada, http://laws.justice.gc.ca/en/c-46/42686.html Canadian Copyright Act, http://laws.justice.gc.ca/en/C-42/index.html

PART TWO – FORENSIC ANALYSIS OF A SYSTEM

Introduction

This paper discusses the investigation conducted by the author in late July-August 2002 as part of the incident response process of the Information Security group of a University. For the purposes of this assignment, significant additional research and expansion has been done to further explore some of the technical details of this case.

Synopsis of Case Facts

Incident Background and Response Measures

Security incidents on a large and active University network are sometimes difficult to spot due to the diverse and unpredictable nature of network traffic considered “normal” in a University environment. This incident investigation resulted from unusual network activity logged by the University intrusion detection system (IDS). The IDS uses Snort as the detection engine, and has visibility of all network traffic between the Internet gateway router and the core switches via a tap device installed on the wire.

The University has an “open network” policy, meaning all network traffic is allowed unless specifically denied, and there is no gateway firewall. This results in a high volume of inbound attacks that penetrate the perimeter and would be detected by the IDS. The vast majority of these attacks are ineffective and responding to all of them would be impossibly time-consuming and pointless in general. Experience has shown that some of the most useful IDS signatures in this environment are those which have been inverted so they look for *outbound* attacks – usually indicating a compromised system. Responding to these alerts can help prevent a single intrusion from being used as a springboard for internal attacks. This incident investigation was started as a result of an alert on one such rule: near midnight, July 26th, 2002, an alarm event was recorded regarding access from outside the University to an SSH (Secure Shell) daemon running on a non-standard port.

The following morning, a network scan was done of the system to confirm the existence of the rogue SSH daemon and a notification was sent to the technical contact, in this case the I.T. support group for the faculty of science. They in turn were able to determine the system owner and reported this to the central I.T. security group. At this point, as they were first on the scene and did not heed the request to not interact with the suspect system, they attempted several impromptu investigative searches and general poking around the system before the security investigator could get there to look at the system systematically. Due to the additional activity, the investigation was made substantially more difficult, particularly in relation to file activity timelines.

After I.T. security arrived, a brief interview with the system owner quickly determined that the rogue SSH daemon was indeed not intentional and the system was likely compromised. The system was an SGI Indy workstation running IRIX, the SGI flavour of UNIX. Its current primary purpose was for reading email and web browsing, and was not deemed essential.

Typical first-responder activity of a compromised system would include recording output of run-time status commands such as *ps*, but unfortunately the local technical support had already rebooted the system into a diagnostic mode known as the IRIX miniroot.

Rather than continuing the on-site investigation using the limited resources available, the system was seized in order to perform a more detailed analysis with the security group's equipment.

System Placement Within the Network

This system was placed on the main University academic network, meaning there were limited network-based access controls to prevent attacks on the system from outside the University network. A rough sketch of the network layout is below – the target system under investigation is listed as “Desktop PC”.

© SANS Institute 2005, Author retains full rights

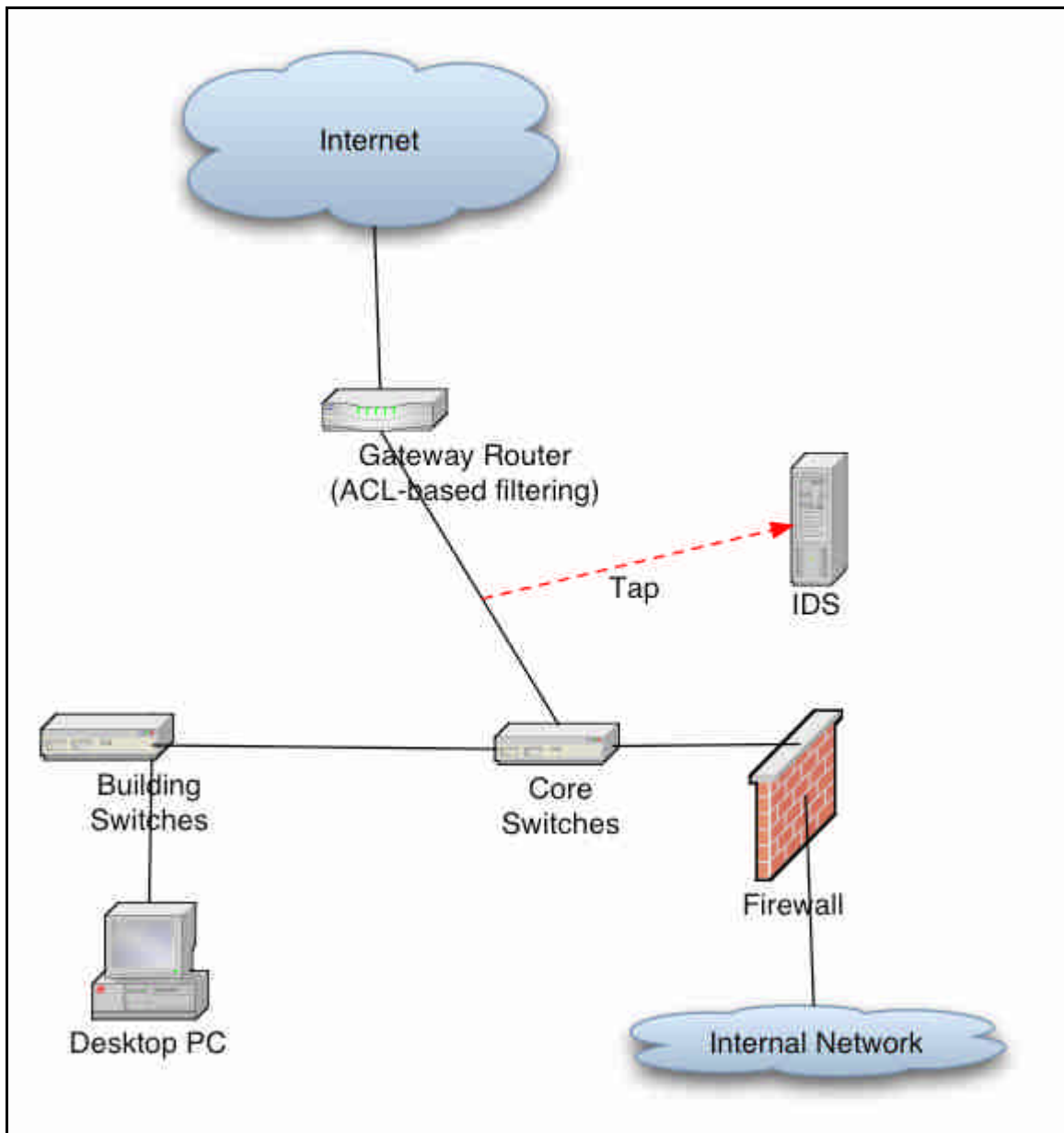


Figure 7 - Network Placement of Compromised PC

As this diagram shows, no firewall device is between the target system and the Internet. That said, certain ports and protocols were filtered at the gateway router using ACLs (access control lists) to provide some protection against common vulnerabilities. According to the known configuration at the time of this incident the ports blocked inbound and outbound in this manner included Windows networking protocol-related ports and a few specific others such as LPR (UNIX printing protocol). However, no ACLs would have affected the most common services and vulnerabilities that may have been present on the target UNIX system. Ports and protocols that were fully accessible from anywhere in the world, each containing well-known potential vulnerabilities include:

- Remote UNIX command-line access protocols:
 - Telnet
 - RSH, Rlogin, Rexec (Remote Shell, Login, Execute)
 - SSH (Secure Shell)
- FTP (File Transfer Protocol)
- RPC (Remote Procedure Call)
- NFS (Network File System)

At this time, an intrusion detection system was deployed by tapping into the Ethernet cable between the core switches and the gateway router, and therefore had visibility of all network traffic inbound and outbound to the Internet.

Regardless of this, we need to consider all possible vulnerable points of entry to the system as the intrusion could have occurred from within the University network itself, where there are no access controls or other restrictions on network activity.

Target System Description

The system is a 1994-era SGI Indy workstation.

It is running IRIX, SGI's proprietary UNIX flavour, version 6.5.3 according to the uname output:

```
# uname -aR
IRIX hostname 6.5 6.5.3m 10151452 IP22
```

The system has one internal hard disk 1GB in size, and an external 2GB hard disk as well as an external CDROM unit. The main unit, external disk, external CDROM, keyboard, and mouse were seized for further investigation.

The system was used to a greater extent at one time as a file server for some other workstations in the physics department, although that function no longer appears in current use. The only user is the owning professor who uses it mainly for email and web browsing.

A network scan of the system before it was disconnected showed it was running the full range of services normally present in a default IRIX installation. This includes RPC (needed for NFS), telnet, ftp, etc. Most of these services are not needed for current use, although the telnet and ftp services are occasionally used by the professor when he is away and needs to access data on the system. The professor has not heard of SSH or the security problems inherent with telnet and FTP. Notably, SSH was never installed on the system, so there was no legitimate SSH daemon at all.

Hardware Seized

The hardware seized is as in the picture below: The main SGI Indy CPU box, including internal hard disk, as well as an external hard disk and enclosure, and external CDROM and enclosure (external enclosures are not pictured).



Chain of Custody

The owning professor volunteered the system to the care of the I.T. security group for the purposes of the investigation and signed a chain of custody form with the following details:

Tag #IS070201	SGI Indy workstation SN#I418508, containing: 64MB RAM, 1GB internal SCSI drive, 3.5 inch floppy drive
Tag #IS070202	Connor CFP1080S 1.06GB SCSI disk SN#184-001062
Tag #IS070203	SGI External SCSI CDROM drive, SN#67846
Tag #IS070204	External SCSI hard drive enclosure containing 2GB disk. OSS ("Open Storage Systems, Inc.") enclosure. SN#108-23

All hardware was taken directly to the I.T. security evidence and investigation area, which is within a locked storage room/office. Two members of the security group carry the only keys. Additionally, the lock to the room is not part of the building master set, therefore the room is not accessible to caretaking staff or others with master keys.

Image Media

Imaging Procedure and Verification

To make a bit-copy of the disk image, the disks are physically removed from the system and attached to the forensic workstation. Since attempting serious analysis of the images within an IRIX environment would be too limiting, the forensics workstation is a Debian Linux system. Fortunately the Linux kernel contains support for the SGI disklabel (partition structure) and both SGI filesystems: EFS and XFS.

It turns out the internal disk was disconnected and unused (apparently bad) – attempts to have the disk be recognized by the SCSI controller failed. The running system was completely located on the external 2GB disk. All further work deals with this external disk.

Step One – Attaching the Disk

Imaging the disk is accomplished by using a SCSI controller attached to the forensics workstation. The read-only (a.k.a. WP, write protect) jumper is set before attaching and booting the forensics system, which boots off an IDE disk. The Linux system recognizes the disk as /dev/sda.

Step Two – Checksum of Original Disk

The SGI disklabel is simplified by Linux to be simply numbered starting from 1. Therefore the single active partition on the disk (not including swap) is /dev/sda1. Before any other action, we use the md5sum command to get the following result for the initial MD5 checksum:

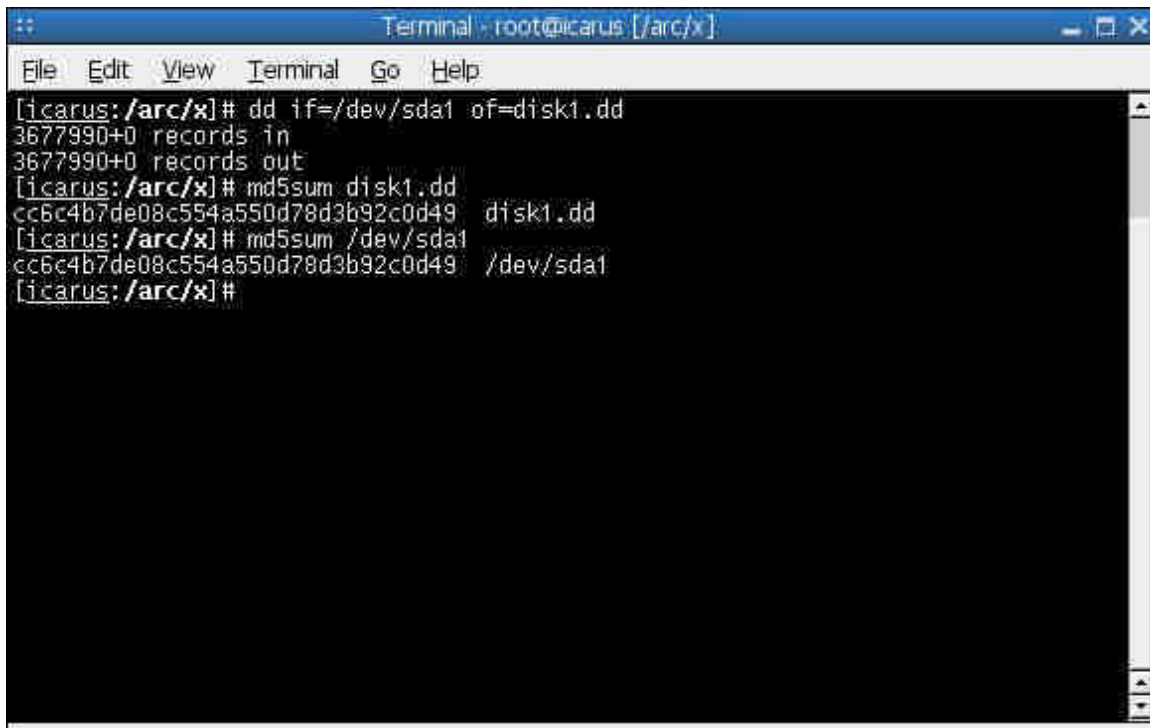
```
[icarus:/arc/x]# md5sum /dev/sda1
cc6c4b7de08c554a550d78d3b92c0d49  /dev/sda1
```

Step Three – Copy Disk Image and Integrity Verification

The partition /dev/sda1 is copied to files by using the UNIX dd command as seen in the screenshot below. The md5sum command is again used on both the image file and the raw partition to show:

- 1) The image file and raw partition have identical checksums, indicating the contents of both are identical.
- 2) The checksum also matches the original raw partition checksum, indicating it has not been changed by the imaging process.

© SANS Institute 2005. Author retains full rights.

A terminal window titled "Terminal - root@icarus [./arc/x]" with a menu bar (File, Edit, View, Terminal, Go, Help). The terminal shows the following commands and output:

```
[icarus:/arc/x]# dd if=/dev/sda1 of=disk1.dd
3677990+0 records in
3677990+0 records out
[icarus:/arc/x]# md5sum disk1.dd
cc6c4b7de08c554a550d78d3b92c0d49  disk1.dd
[icarus:/arc/x]# md5sum /dev/sda1
cc6c4b7de08c554a550d78d3b92c0d49  /dev/sda1
[icarus:/arc/x]#
```

Figure 8 – Imaging/MD5 sum of Media

Note on Swap Space

As mentioned previously, the local system support personnel had booted to the IRIX “miniroot” to do some investigation of their own before the system was seized. Unfortunately, the miniroot creates a temporary bootable system within the *swap area* of the system disk, effectively erasing any evidence which may have been present there. Since the swap area was completely overwritten it was not considered for further analysis.

Media Analysis of System

Tools

The forensics workstation is a Debian Linux system. Some of the work must be done on an actual IRIX system as well, and fortunately we have access to such a system other than the one we are investigating. Most of the analysis is done with typical UNIX file tools as well as some IRIX-specific commands. Detailed timeline analysis is done with elements of The Coroners Toolkit¹.

Unfortunately, there are few forensic tools available for working with XFS filesystems. Even the ability to read XFS volumes requires an SGI IRIX system, or a recent Linux system with XFS compiled into the kernel. Linux has many more useful tools in general and makes a better platform for examining XFS than even a real IRIX system. Due to this lack of tools which understand the XFS filesystem, we are forced to perform the analysis

¹ The Coroner’s Toolkit (TCT), <http://www.porcupine.org/forensics/tct.html>

on a mounted volume. Fortunately, this is extremely simple in Linux as it can perform “loopback” mounts of the disk image file without modifying the contents in any way.

The command below is used to mount the disk image on /target:

```
mount -o ro,loop,noatime,noexec,nodev sdal_xfs.dd /target
```

The meaning of the mount command options are as follows:

- ro = read-only, prohibit any modification of the contents
- loop = this is a loopback mount (e.g. treat file as if it were raw disk)
- noatime = do not update access times, redundant with ro, but force of habit
- noexec = do not permit execute permission bit to be set on any file, to avoid accidentally running programs from the disk image. Despite the fact the target is IRIX and the forensic workstation host is Linux, this is not redundant since shell or perl-language scripts would likely work just as well in either operating system.
- nodev = ignore the special meaning of device inodes; treat them as normal files

Comparison with Original Operating System Software

A method of determining if any of the operating system programs and/or libraries have changed after an intrusion is to perform checksums of the files and compare those with known good originals. In this case, we do not have an original copy of the version of IRIX used to install this system, however there are some basic tools within the IRIX package management system which can give us some information about this kind of change.

Using Native Package Management Tools

“Inst” is the IRIX native software/package management system. All the core operating system files are part of an inst software package. The inst program itself keeps a database of what files belong to what package and also keeps some integrity information such as checksums of the original files which we can use. Other programs related to inst include “versions” which shows what packages are installed on the system. A lower-level command is “showfiles” (usually invoked by inst rather than run on its own) which is the easiest way to get a list of modified files. An example of using this command for this purpose is:

```
showfiles ! -c -a ! -w -a -m
```

This cryptic command is how you tell showfiles to list:

- 1) Files that are NOT (!) configuration files (-c)
- 2) AND (-a) NOT (!) files installed by an overlay package (-w)
- 3) AND (-a) files that have been modified from the checksum stored in the package database (-m)

We exclude configuration files since it is normal for them to be modified from the originals. Also we exclude “overlay” files – overlays are special package updates, typically for security and maintenance patches which are unfortunately not recorded in

the checksum database, therefore files installed as part of an overlay will show up as being modified from the original. It will not be possible to tell if those files have been modified by the overlay or due to the intrusion using this command alone. The resultant output of this command will be any files from base packages that have been modified or deleted.

Since IRIX cannot mount disk image files as Linux can, and Linux does not have the “inst” utility, we have to do a trick to make the mounted volume within Linux available to the IRIX system. One way to do this is by making use of NFS, the Unix Network File System. After exporting the mounted volume (read-only) to the IRIX system, we login to the “secure” IRIX system, mount the NFS volume in /mnt.

Next, we need to tell the “showfiles” command to operate as if /mnt were the real root directory; fortunately this is simply accomplished with the -r option as follows:

```
# showfiles ! -c -a ! -w -a -m -r /mnt
f 49244      3 pcnfsd.sw.base          etc/config/pcnfsd
f 47386 43412 eoe.sw.base          sbin/ps
f 64381 69656 eoe.sw.base          usr/lib/iaf/scheme
#
```

It appears only a couple files were modified from their original form which cannot be accounted for by an operating system patch.

Examination of /etc

The /sbin/ps file is very suspicious, but first we check a few important system files to get an idea of the prior use of the system.

```
[callisto:/mnt]# cat etc/fstab
/dev/root / xfs rw,raw=/dev/rroot 0 0
z:/d/zoo /d/zoo nfs rw,soft,bg 0 0
z:/d/scr1 /d/scr1 nfs rw,soft,bg 0 0
b:/d/b /d/b nfs rw,soft,bg 0 0
b:/d/zs /d/zs nfs rw,soft,bg 0 0
b:/d/scr2 /d/scr2 nfs rw,soft,bg 0 0
b:/d/local /d/local nfs rw,soft,bg 0 0
b:/d/scr6 /d/scr6 nfs rw,soft,bg 0 0
```

Several data directories from other UNIX systems are mounted via NFS, including the home directories of several users. These will need to be checked for unauthorized activity in a separate investigation.

A check of /etc/passwd, /etc/shadow, and /etc/group show no extra or unusual accounts on the system.

Start-up Files and Processes

The possible ways of starting a program at system start time in IRIX are similar to most other UNIX flavours:

- 1) /etc/inittab – config for init

This file mainly contains lines for “getty” which is the terminal login prompter. Typically this file is seldom changed and comparison with the inittab from the “secure” IRIX system shows no significant changes.

2) /etc/init.d – startup scripts

These startup scripts (mostly one per service) can be quite complex and it is difficult to tell if anything is added. Checking with the UNIX tool “diff” with “-b -r” will show any changes to files from the secure IRIX system and the image and list the changes with the filenames involved:

```
# diff -b -r /etc/init.d /mnt/etc/init.d
diff -b -r /etc/init.d/xfsd /mnt/etc/init.d/xfsd
60a61
>/dev/pts/01/sshd
```

Therefore, line 61 has been added to the xfsd startup script on the compromised system, and it appears to be starting an SSH daemon from an unusual location. We investigate this further below.

3) /etc/rc* directories – links to startup scripts in init.d, however files containing scripts could also be placed here and have the same effect.

All the files in these directories are symlinks to files within /etc/init.d (as they should be) and therefore there is no need to further investigate. The find command is the easy way to verify this:

```
# find /etc/rc* -type f
/etc/rc0
/etc/rc2
/etc/rc3
#
```

These files are short scripts which run every script found within the corresponding rc directory.

4) inet.d – inet config file; many network services are started via inet

This file does not appear to have been modified from the initial install and contains a number of services, while not backdoors in themselves they are security risks.

```
telnet  stream tcp      nowait  root    /usr/local/etc/tcpd
/usr/etc/telnetd
shell  stream tcp      nowait  root    /usr/local/etc/tcpd
/usr/etc/rshd -L
login  stream tcp      nowait  root    /usr/local/etc/tcpd
/usr/etc/rlogind
exec   stream tcp      nowait  root    /usr/local/etc/tcpd
/usr/etc/rexecd
```

In particular, IRIX telnet has a long history of security problems.

5) chkconfig – on IRIX this is another method of configuring startup services, although really it is just a wrapper for some init.d scripts.

Chkconfig -s lists all settings:

```

# chkconfig -s
Flag          State
=====
acct          off
cleanpowerdown on
cpumeter      on
esp          on
lockd        on
lp           on
mkpd         on
mmscd        on
network      on
nfs          on
nsd          on
numastatd    on
pmcd         on
privileges   on
quotas       on
routed       on
rtmond       on
sar          on
savecore     on
sendmail_cf  on
sesdaemon    on
sysevent     on
ts           on
verbose      on
vswap        on
xlv          on
xntp         on
autoconfig_ipaddress off
autofs       off
automount    off
desktop     off
fcagent      off
fontserver   off
gated        off
ipaliases    off
mediad       off
miser        off
mrouted      off
named        off
nds          off
noiconlogin  off
nostickytmp  off
nss_fasttrack off
pcnfsd       off
pmie         off
proclaim_relayagent off
proclaim_server off
proxymngr    off
quickpage    off
quotacheck   off
rarpd        off
rsvpd        off
rwhod        off

```

sdpd	off
sendmail	off
sgi_apache	off
snetd	off
soundscheme	off
timed	off
timeslave	off
videod	off
visuallogin	off
webface	off
webface_apache	off
windowsystem	off
xdm	off
yp	off
ypmaster	off
ypserv	off

Options for all these are stored in the /etc/config directory and there is nothing unusual there. It appears that only one non-configuration file has been modified from the base install: pcnfsd. At some point after the initial install, the pcnfsd service was turned off, but this is not unusual or unexpected.

- 6) cron – although not a normal way of configuring a start-up program, a cron job which checks periodically to see if a process is running, and if not starts it, would have a similar effect as using the normal startup scripts, and has the advantage of being something commonly overlooked.

/var/spool/cron contains the configuration for cron. There are only entries for 4 system users: root, sys, diag, adm. These start periodic maintenance scripts and there does not appear to be anything out of the ordinary. Checking these 4 files with the “secure” IRIX system shows no modifications.

Setuid/Setgid Files

Using the find(1) command we can investigate for suspicious setuid/setgid files.

Checking for files with the octal permission bit 4000 set will show those with setuid permission, meaning files which execute with the privileges of the user which owns the file, NOT the user executing the file as would normally occur. This can allow privilege escalation if there are bugs in the program, and it is also an easy way to hide a back-door program. Setgid files are similar but provide privileges of the owning group, not the user - the interesting results are shown:

```
#find . -type f \( -perm -4000 -o -perm -2000 \) -ls
217715  43 -rwxr-sr-x   1 root    root      43632 Nov 24  2001
./dev/pts/01/backup/ps
217714  68 -rwsr-xr-x   1 root    root      69940 Nov 24  2001
./dev/pts/01/backup/scheme
7411610 43 -rwxr-sr-x   1 root    root      43632 Nov 24  2001
./lib/ldlibps.so
434077  43 -rwxr-sr-x   1 root    root      43632 Nov 24  2001
./sbin/ps
```

It appears the ‘ps’ command has been changed and is setgid-root, meaning whoever runs it will run the program with root group privileges – this is not as powerful as root-user

privileges. The 'ps' command was also seen in the comparison with the IRIX package system checksums and is not the original file. Scheme was also modified and it too shows up here, and in this case it does have setuid-root to run as the root user when executed.

Note that 3 files have the same file size. An md5sum comparison:

```
md5sum lib/ldlibps.so sbin/ps dev/pts/01/backup/ps
06ba62f700962377a8f0ce4c1d6410cb lib/ldlibps.so
06ba62f700962377a8f0ce4c1d6410cb sbin/ps
787417596e47be5c312a455c046e12c2 dev/pts/01/backup/ps
```

Interestingly, /lib/ldlibps.so does not belong in the system at all, and an md5sum shows it is the same as /sbin/ps. The copy of ps in the "backup" directory may be the original. The purpose of ldlibps.so is unclear – it is not a shared-object library. It seems likely to just be another copy of the replacement ps in case the original is removed.

Timeline Analysis

System Installation and Updates

The original install date is likely to have been Apr 10 1999 as most directories and symlinks appear to have this timestamp. Another more direct way of determining this is to use the IRIX "versions" command to see the install date of the core system software:

```
# versions eoe.sw.base
I = Installed, R = Removed
```

	Name	Date	Description
I	eoe	04/10/1999	IRIX Execution Environment, 6.5.3m
I	eoe.sw	04/10/1999	IRIX Execution Environment Software
I	eoe.sw.base	04/10/1999	IRIX Base Execution Environment

Of course, the accuracy of this information presumes that the date and time were set correctly on the computer's clock at install time. In any case it is likely that any file with this timestamp was part of the original installation, such as /etc/uucp (to pick an example).

```
[callisto:/mnt]# find . -newer etc/uucp | wc -l
807
```

To get an idea of the extent of modifications since that time, 807 files on the entire system (excluding home directories since those are on the internal disk) have been modified since the original install date. On further investigation, it appears most of these are in the root directory as a result of the user logging into and using the system as the root user. Using Netscape as the root user creates an entire /.netscape/cache structure along with user rc-files. There are no apparent updates to the system after the initial installation.

To get a general idea of where changes have occurred, we look for modified directories using find (option: -type d):

```
[callisto:/mnt]# find . -type d -newer etc/uucp
.
```

```

./dev
./dev/pts
./dev/pts/01
./dev/pts/01/bnc
./dev/pts/01/etc
./dev/pts/01/backup
./dev/xlv
./dev/rxlv
./dev/ttyd0
./dev/ttyd0/...
./dev/ttyd0/.../.old
./dev/ttyd0/.../.old/.up
./dev/ttyd0/.../.old/obj
./dev/ttyd0/.../.old/src
./dev/input
./etc
./usr/bin/.sh_history
[listing truncated]

```

Immediately we find two groups of suspicious directories buried in /dev, which normally do not change much from system installation. /dev is a known favorite place for intruders to hide their files and this is clearly the case here. First, taking a look at /dev/pts and /dev/ttyd0:

```

[callisto:/mnt]# ls -la dev/pts
total 10
 0 drwxr-xr-x   3 root    root          142 Nov 24  2001 ./
10 drwxr-xr-x  16 root    root          9728 Oct  2 15:47 ../
 0 crw--w--w-   2 269    dialout    15,   0 Jul 26 03:14 0
 0 drwxr-xr-x   5 root    root          132 Jul 26 03:11 01/
 0 crw--w--w-   2 269    dialout    15,   1 May  6  2002 1
 0 crw--w--w-   2 root    root          15,  10 Mar 28  1996 10
 0 crw-----   2 root    root          15,  11 Mar 28  1996 11
 0 crw--w--w-   2 269    dialout    15,   2 May  3  1999 2
 0 crw-rw-rw-   2 root    root          15,   3 Apr  2  1999 3
 0 crw-rw-rw-   2 root    root          15,   4 Oct 21  1999 4
 0 crw--w--w-   2 root    root          15,   5 Mar  4  1997 5
 0 crw--w--w-   2 root    root          15,   6 Dec 17  1996 6
 0 crw--w--w-   2 root    root          15,   7 Apr 18  1996 7
 0 crw--w--w-   2 root    root          15,   8 Apr 18  1996 8
 0 crw--w--w-   2 root    root          15,   9 Mar 25  1996 9

[callisto:/mnt]# ls -la dev/ttyd0
total 10
 0 drwxr-xr-x   3 root    bin           21 Nov 26  2001 ./
10 drwxr-xr-x  16 root    root          9728 Oct  2 15:47 ../
 0 drwxr-xr-x   3 root    bin           22 Nov 26  2001 .../

```

Since the contents of /dev/pts is supposed to be device nodes, the presence of a directory called /dev/pts/01 is clearly suspicious. Also /dev/ttyd0 is equally suspicious since it is a directory and not a device node. It contains a subdirectory named '...', which is named to be a hidden file which could be easily overlooked. The contents of both these directories is worth investigating further, but before going on we note that they have much different timestamps - almost 9 months apart starting in November 2001.

MAC Time Analysis

We use the grave-robber tool from TCT to walk the filesystem and gather MAC times for analysis. This tool can gather MAC times from a mounted filesystem, which is the only way we can gather this information in the absence of forensics tools that understand XFS block-level structures.

```
# cd /tmp
# grave-robber -m /mnt
# mactime -b /test/tct/tct-1.11/data/callisto/body 11/01/2001-
07/27/2002 | less
```

This should show the activity during the original compromise and more recent intruder activity.

```
Sat Nov 24 2001 10:09:32      65536 m.. -rw----- 0      0
2836619 /mnt/lost+found/2836619
Sat Nov 24 2001 10:10:28      142 m.c drwxr-xr-x 0      0
5243319 /mnt/dev/pts
Sat Nov 24 2001 10:10:29      69940 m.c -rwsr-xr-x 0      0
217714 /mnt/dev/pts/01/backup/scheme
Sat Nov 24 2001 10:10:30      43632 m.c -rwxr-Sr-x 0      0
217715 /mnt/dev/pts/01/backup/ps
69940 m.c -rwsr-xr-x 0      0
2097573 /mnt/usr/bin/login
10240 m.c -rw-r--r-- 0      0
1068055 /mnt/dev/pts/01/etc/etc.tar
Sat Nov 24 2001 10:10:31      43632 mac -rwxr-Sr-x 0      0
7411610 /mnt/lib/ldlibps.so
128 m.c drwxr-xr-x 0      0
7340408 /mnt/lib
54 m.c drwxr-xr-x 0      0
217710 /mnt/dev/pts/01/backup
Sat Nov 24 2001 10:10:33      45424 m.c -rwxr-xr-x 0      0
6323529 /mnt/usr/etc/telnetd
Sat Nov 24 2001 10:10:34      65536 .ac -rw----- 0      0
2836619 /mnt/lost+found/2836619
Sat Nov 24 2001 10:11:10      43632 m.c -rwxr-Sr-x 0      0
5434077 /mnt/sbin/ps
```

The usage habits of the system owner were to only log in during weekdays during business hours, and usually only to read email. Therefore, these entries are suspicious. We can clearly see the intrusion time when suddenly a number of files are created and a few system binaries are replaced; notably, /usr/etc/telnetd.

```
Mon Nov 26 2001 09:36:02      10 .a. lrwxr-xr-x 0      0
1048947 /mnt/usr/adm
15 .a. lrwxr-xr-x 0      0
1048928 /mnt/usr/preserve
Mon Nov 26 2001 09:41:51      21 m.c drwxr-xr-x 0      2
2962639 /mnt/dev/ttyd0
Mon Nov 26 2001 09:42:01      22 m.c drwxr-xr-x 0      2
3163085 /mnt/dev/ttyd0/...
Mon Nov 26 2001 09:45:03      16781 m.. -rw-r--r-- 0      2
4310960 /mnt/dev/ttyd0/.../old/
Mon Nov 26 2001 09:47:49      22260 .ac -rw-r--r-- 0      0
6809343 /mnt/dev/ttyd0/.../old/obj/dccchat.o
```

```

Mon Nov 26 2001 09:47:50      16781 ..c -rw-r--r-- 0      2
4310960 /mnt/dev/ttyd0/.../.old/
Mon Nov 26 2001 09:48:17     353876 ..c -rwxr-xr-x 0      0
4342497 /mnt/dev/ttyd0/.../.old/tcsh
Mon Nov 26 2001 09:48:23         4 .a. -rw----- 0      2
4310964 /mnt/dev/ttyd0/.../.old/divx.ignl
Mon Nov 26 2001 09:48:25     16781 .a. -rw-r--r-- 0      2
4310960 /mnt/dev/ttyd0/.../.old/
                                87137 .a. -rw----- 0      2
4310965 /mnt/dev/ttyd0/.../.old/divx.log
Mon Nov 26 2001 09:48:26         5 mac -rw----- 0      2
4310967 /mnt/dev/ttyd0/.../.old/divx.pid
Mon Nov 26 2001 09:49:20     29516 .a. -r--r--r-- 0      0
6665401 /mnt/usr/lib/libcrypt.so
Tue Nov 27 2001 07:45:00     353876 .a. -rwxr-xr-x 0      0
4342497 /mnt/dev/ttyd0/.../.old/tcsh
Tue Nov 27 2001 07:46:36         9 m.c drwxr-xr-x 0      2
5335617 /mnt/dev/ttyd0/.../.old/.up
Tue Nov 27 2001 14:37:29     1409 .a. -rw----- 0      2
4310962 /mnt/dev/ttyd0/.../.old/divx.msg
Thu Nov 29 2001 13:29:17     1409 m.c -rw----- 0      2
4310962 /mnt/dev/ttyd0/.../.old/divx.msg

```

Here is more intruder activity. Although there is little evidence available now to tell what actually occurred, some hints such as an access of libcrypt.so are present. Also noted is a reference to "DivX" which is a video codec commonly used for distributing pirated movies on the Internet.

```

Fri Nov 30 2001 10:15:02     87137 m.c -rw----- 0      2
4310965 /mnt/dev/ttyd0/.../.old/divx.log
Fri Nov 30 2001 10:21:36      228 ma. -rw----- 0      2
4310966 /mnt/dev/ttyd0/.../.old/divx.xdcc.bkup
Fri Nov 30 2001 10:51:37      228 ..c -rw----- 0      2
4310966 /mnt/dev/ttyd0/.../.old/divx.xdcc.bkup
                                228 mac -rw----- 0      2
4342510 /mnt/dev/ttyd0/.../.old/divx.xdcc
                                4 m.c -rw----- 0      2
4310964 /mnt/dev/ttyd0/.../.old/divx.ignl
                                512 m.c drwxr-xr-x 0      2
4196778 /mnt/dev/ttyd0/.../.old

```

The intruder is back, and attempting dcc transfers of divx material from the looks of these filenames. There is a long break of several months before we have evidence of the subject returning. In this case it is to install an IRC tool, bnc.

```

Wed Jul 24 2002 21:21:40     1536 ..c drwxr-xr-x 1007    100
6880055 /mnt/dev/pts/01/bnc
Sat Feb 09 2002 19:42:02    568336 m.. -rwxr-xr-x 1007    100
5435096 /mnt/dev/pts/01/pico
Sat Feb 09 2002 19:42:05     26052 m.. -rwxr-xr-x 1007    100
5435098 /mnt/dev/pts/01/identd
Sat Feb 09 2002 19:42:53        82 m.. -rwxr-xr-x 1007    100
5435094 /mnt/dev/pts/01/bnc.conf
Sat Feb 09 2002 20:59:23     5307 ..c -rw-r--r-- 1007    100
6880056 /mnt/dev/pts/01/bnc/CHANGES
                                18154 ..c -rw-r--r-- 1007    100
6880057 /mnt/dev/pts/01/bnc/server.c

```

```

          914 ..c -rw-r--r-- 1007      100
6880058 /mnt/dev/pts/01/bnc/Makefile
Sat Feb 09 2002 20:59:24      5760 ..c -rw-r--r-- 1007      100
6880251 /mnt/dev/pts/01/bnc/bnc.o

```

Finally, there is one more incursion the next day:

```

Thu Jul 25 2002 18:10:31      87564 .a. -rwxr-xr-x 0          0
6297636 /mnt/usr/etc/ftpd
Fri Jul 26 2002 00:01:02      13932 .a. -r-x----- 0          0
7593048 /mnt/usr/sbin/chkulent
Fri Jul 26 2002 03:08:57      85352 .a. -rwxr-xr-x 0          0
7398444 /mnt/usr/sbin/gzip
Fri Jul 26 2002 03:09:51      2218 .a. -r--r--r-- 0          0
4259038 /mnt/usr/include/sys/ioccom.h
          22173 .a. -r--r--r-- 0          0
4212661 /mnt/usr/include/sys/socket.h
          3049 .a. -r--r--r-- 0          0
4212701 /mnt/usr/include/sys/termio.h
          2537 .a. -r--r--r-- 0          0
107307 /mnt/usr/include/fcntl.h
          2488 .a. -r--r--r-- 0          0
4212681 /mnt/usr/include/sys/ttydev.h
          1655 .a. -r--r--r-- 0          0
4212577 /mnt/usr/include/sys/ioctl.h
          10561 .a. -r--r--r-- 0          0
4259025 /mnt/usr/include/sys/fcntl.h
          3133 .a. -r--r--r-- 0          0
4212535 /mnt/usr/include/sys/file.h
          7470 .a. -r--r--r-- 0          0
6668618 /mnt/usr/include/net/soioctl.h
          15896 .a. -r--r--r-- 0          0
4212702 /mnt/usr/include/sys/termios.h
...
Fri Jul 26 2002 03:11:31      132 m.c drwxr-xr-x 0          0
7411557 /mnt/dev/pts/01
Fri Jul 26 2002 03:12:01      660 .a. -rw----- 0          0
6654787 /mnt/.ssh/known_hosts
Fri Jul 26 2002 03:12:04      512 mac -rw----- 0          0
6351951 /mnt/.ssh/random_seed
          660 m.c -rw----- 0          0
6654787 /mnt/.ssh/known_hosts
Fri Jul 26 2002 03:14:37      0 m.. crw--w--w- 269      20
5243340 /mnt/dev/ttyq0
Fri Jul 26 2002 03:14:41      2296 m.c -rw----- 0          2
2962641 /mnt/usr/bin/.sh_history
          0 .a. crw--w--w- 269      20
5243340 /mnt/dev/ttyq0
Fri Jul 26 2002 03:53:25      512 m.c -rw----- 0          0
1086229 /mnt/dev/pts/01/etc/ssh_random_seed

```

These entries show an access to gzip and a number of C-header files, which is probably a result of the intruder uncompressing and compiling source code - it is certainly not the legitimate user doing this at 3AM. We can confirm it from the .sh_history file noted below. Finally, we can see when the SSH backdoor is installed by the creation of the ssh_random_seed file in /dev. The IDS also notices a login to the SSH backdoor

(probably a test to make sure it worked) at around this time and the two events flag the system as a likely compromise:

```
07/26-03:54:02.235567  [**] [1:2000354:1] BACKDOOR Non-Standard SSH
Port Usage [**] [Priority: 0] {TCP} a.a.a.a:13000 -> b.b.b.b:38120
07/26-03:54:02.236054  [**] [1:2000354:1] BACKDOOR Non-Standard SSH
Port Usage [**] [Priority: 0] {TCP} b.b.b.b:38120 -> a.a.a.a:13000
```

Later that morning, due to the IDS alerts, the system is taken off the network. You can then see a local user or admin scanning the system looking for signs of intrusion.

```
Fri Jul 26 2002 12:20:30      11 .a. lrwxr-xr-x 0      0
7340169 /mnt/usr/sbin/perl      7 .a. lrwxr-xr-x 0      0
7340168 /mnt/usr/sbin/pathconf    9 .a. lrwxr-xr-x 0      0
7447672 /mnt/usr/sbin/aview      20 .a. lrwxr-xr-x 0      0
7750325 /mnt/usr/sbin/ttmv       9 .a. lrwxr-xr-x 0      0
7699138 /mnt/usr/sbin/u64check   26 .a. lrwxr-xr-x 0      0
7826575 /mnt/usr/sbin/cvd       30 .a. lrwxr-xr-x 0      0
7826562 /mnt/usr/sbin/cvquery    6 .a. lrwxr-xr-x 0      0
7567596 /mnt/usr/sbin/gzcmp      19 .a. lrwxr-xr-x 0      0
7398479 /mnt/usr/sbin/inform     30 .a. lrwxr-xr-x 0      0
7826567 /mnt/usr/sbin/cvusage    8 .a. lrwxr-xr-x 0      0
7824224 /mnt/usr/sbin/cdman     10 .a. lrwxr-xr-x 0      0
7640643 /mnt/usr/sbin/espro
```

This continues for every file in /bin, /usr/bin, /sbin, and others, destroying any further evidence of intruder activity. Immediately after this, the system is seized.

Recover Deleted Files

Unfortunately, since the filesystem is XFS, recovering deleted files is very difficult. There are numerous sources of information on the Internet which all lead to the conclusion that the design of the XFS filesystem does not make deleted file recovery feasible, as summarized in the XFS FAQ on the SGI web site:

Does the filesystem have a undelete function?

There is no undelete in XFS, in fact once you delete something, the chances are the space it used to occupy is the first thing reused. Undelete is really something you have to design in from the start. Getting anything back after an accidental `rm -rf` is near to impossible.

In addition to the likelihood of the content being overwritten, finding the content itself is much more difficult than with a filesystem such as EXT2. In EXT2, files can be recovered by first locating the inode which points to the data. Although the inode is

unlinked from any directory, it will usually still be intact once found, and then the data can be recovered from that point if it hasn't been overwritten. In XFS, finding the inode may not be any more of a problem, but due to the way XFS works, it will not contain any reference to the location of the data if the file was deleted. A more technical description from an SGI engineer is in this message,
http://oss.sgi.com/projects/xfstest/mail_archive/200202/msg00013.html:

Unfortunately when we free an inode we also remove all the extents from it. So even if those extents used to be in the inode, they are not there now. The reason this happens is that deleting a file is an unbounded operation in terms of how complex it is, so it has to be broken up into lots of separate transactions - and between each transaction the inode needs to be 'correct' so that if we crash in the middle of removing a file we can complete the remove during recovery. So when we unlink a file there is a transaction to put it into a linked list of files to be removed during recovery. Then when its reference count drops to zero we do a sequence of transactions which remove 2 extents each, then finally we do a last transaction to mark the inode free and remove it from the list.

Since we never free the inode blocks at all, the inode is definitely still there, there is just not very much in it anymore.

In addition to the added difficulty in finding the deleted data without anything pointing to it, XFS allocates space based on free blocks placed in what is essentially a LIFO (last in, first out) queue. This means that recently-freed blocks will tend to be the first blocks used for any new allocations.

As an example of this, we create a blank XFS filesystem, create and then delete a file, and then create a new file and see if it is possible to restore the original.

```
# dd if=/dev/zero of=test_xfs bs=1024 count=65535
65535+0 records in
65535+0 records out
67107840 bytes transferred in 1.468111 seconds (45710329 bytes/sec)
# mkfs -t xfs test_xfs
meta-data=test_xfs          isize=256      agcount=3, agsize=4096
blks
=
=
data      =                  sectsz=512
=                  bsize=4096   blocks=12288, imaxpct=25
=                  sunit=0      swidth=0 blks,
unwritten=1
naming    =version 2         bsize=4096
log       =internal log     bsize=4096   blocks=1200, version=1
=                  sectsz=512   sunit=0 blks
realtime  =none             extsz=65536   blocks=0, rtextents=0
```

A test 64MB XFS filesystem has been created. We mount this as a loopback device and then create a single 1K file on it with a content of all "A"s to make it simple to search for it.

```
# mount -o loop,rw -t xfs test_xfs /mnt
# ls /mnt
total 0
# perl -e 'print "A"x1024' > /mnt/testfile
```

ne with all “B”s :

ne with all “B”s

```
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
```

```
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB
```

BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB

```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
# strings test_xfs | grep AAAA
#

```

No results were returned by the final grep for AAAA, indicating that the data has been overwritten.

String Search

Keywords and Search Procedure

Performing a string search on the media can help find interesting bits of information. Some typical things to look for:

- 1) Environment variables. This is useful for investigating the memory space of running processes from swap which we do not have access to.
- 2) Password strings. username:password:uid combinations which in UNIX are often expressed in colon-separated form with an alphabetic name/password, some encrypted password string and numeric UID.

```
strings root.dd | egrep -A3 '[a-zA-Z]+:[\*0-9]+'

```

```

guest*:10878::::::
daemon;
daemon*:10878::::::
root;
root:tw3ciLDkldyCk:10878::::::

```

- 3) Syslog entries. Searching for login events is possible as with the pattern below (egrep or perl which has more flexible regular expressions) shows some interesting entries.

```

# strings root.dd | egrep -A3 'sshd\[([0-9]+)\]'
Jul 19 01:11:32 6W:wildflag sshd[20771]: log: Connection from
194.93.79.142 port 3421
Jul 19 01:11:32 6W:wildflag sshd[20771]: log: Could not reverse
map address 194.93.79.142.
Jul 19 01:11:33 6W:wildflag sshd[20771]: fatal: Did not receive
ident string.
Jul 19 01:20:41 6W:wildflag sshd[20815]: log: Connection from
204.252.127.148 port 3444
Jul 19 01:20:49 6W:wildflag sshd[20815]: log: reverse mapping
checking gethostbyname for tiger.ispalliance.net failed -
POSSIBLE BREAKIN ATTEMPT!

```

```

strings root.dd | perl -n -e 'print if /sshd\[([0-9]+)\]/'
Jul 19 01:20:52 3W:wildflag sshd[20661]: fatal: Local: Corrupted
check bytes on input.
Jul 19 01:20:55 3W:wildflag sshd[20680]: fatal: Local: Corrupted
check bytes on input.
Jul 19 01:20:58 3W:wildflag sshd[20823]: fatal: Local: Corrupted
check bytes on input.

```

```
Jul 19 01:21:01 3W:wildflag sshd[20810]: fatal: Local: Corrupted
check bytes on input.
Jul 19 01:21:03 3W:wildflag sshd[20821]: fatal: Local: Corrupted
check bytes on input.
Jul 19 01:21:06 3W:wildflag sshd[20835]: fatal: Local: Corrupted
check bytes on input.
```

In this case, the system was unsuccessfully attacked by an SSH exploit which was probably written for an i386 Linux architecture, not MIPS IRIX. These log file entries were simply rotated out and were probably not related to the successful intrusion.

- 4) Media Files. Looking for divx turns up several filename which have already been mentioned in the timeline, but none that are new (note that the 'i' at the end of the pattern indicates a case-insensitive search):

```
# strings root.dd | perl -n -e 'print if /DIVX/i'
divx.pid
divx.log
divx.xdcc
```

Conclusions

Synopsis of Intruder Activity

We have located several areas of intrusion activity and relevant timeline. To provide a synopsis of the activity we go back to the repositories left by the intruder(s) in /dev.

```
find dev/ttyd0 -ls | cut -c48-
  21 Nov 26   2001 dev/ttyd0
  22 Nov 26   2001 dev/ttyd0/...
  512 Nov 30   2001 dev/ttyd0/.../.old
16781 Nov 26   2001 dev/ttyd0/.../.old/\ \
    9 Nov 27   2001 dev/ttyd0/.../.old/.up
  512 Nov 26   2001 dev/ttyd0/.../.old/obj
112368 Jan 10   2001 dev/ttyd0/.../.old/obj/iroffer.o
28788 Jan 10   2001 dev/ttyd0/.../.old/obj/transfer.o
132992 Jan 10   2001 dev/ttyd0/.../.old/obj/misc.o
 11880 Jan 10   2001 dev/ttyd0/.../.old/obj/display.o
181068 Jan 10   2001 dev/ttyd0/.../.old/obj/admin.o
 22260 Jan 10   2001 dev/ttyd0/.../.old/obj/dccchat.o
   8188 Jan 10   2001 dev/ttyd0/.../.old/obj/plugins.o
 57752 Jan 10   2001 dev/ttyd0/.../.old/obj/utilities.o
 18328 Jan 10   2001 dev/ttyd0/.../.old/obj/upload.o
   512 Nov 26   2001 dev/ttyd0/.../.old/src
 68693 Dec  5   2000 dev/ttyd0/.../.old/src/iroffer.c
13026 Dec  5   2000 dev/ttyd0/.../.old/src/transfer.c
10074 Dec  5   2000 dev/ttyd0/.../.old/src/headers.h
  3153 Dec  5   2000 dev/ttyd0/.../.old/src/globals.h
  5691 Dec  5   2000 dev/ttyd0/.../.old/src/defines.h
 66727 Dec  5   2000 dev/ttyd0/.../.old/src/misc.c
  3274 Dec  5   2000 dev/ttyd0/.../.old/src/display.c
 86402 Dec  5   2000 dev/ttyd0/.../.old/src/admin.c
  9563 Dec  5   2000 dev/ttyd0/.../.old/src/dccchat.c
  5629 Dec  5   2000 dev/ttyd0/.../.old/src/plugins.c
28175 Dec  5   2000 dev/ttyd0/.../.old/src/utilities.c
```

```

6553 Dec 5 2000 dev/ttyd0/.../.old/src/upload.c
353876 Jan 10 2001 dev/ttyd0/.../.old/tcsh
87137 Nov 30 2001 dev/ttyd0/.../.old/divx.log
1409 Nov 29 2001 dev/ttyd0/.../.old/divx.msg
5 Nov 26 2001 dev/ttyd0/.../.old/divx.pid
4 Nov 30 2001 dev/ttyd0/.../.old/divx.ignl
228 Nov 30 2001 dev/ttyd0/.../.old/divx.xdcc
228 Nov 30 2001 dev/ttyd0/.../.old/divx.xdcc.bkup

```

This looks like the intruder came back several times to set up a DCC file copy site. The log file shows that this was only partially successful, probably because the system has only a few hundred megabytes of available disk. Those DivX movies are too big for this old Indy.

```

** 2001-11-26-14:48:35: Attempting Connecting to dungeon.ircgate.net
6667 (direct)
** 2001-11-26-14:48:35: WARNING: Can't Resolve Server Host
** 2001-11-26-14:48:35: Attempting Connecting to cs.ircgate.net 6667
(direct)
** 2001-11-26-14:48:35: Server Connection Established, Logging In
** 2001-11-26-14:49:00: Joined #DIVX
** 2001-11-26-14:49:19: DCC Chat Requested
** 2001-11-26-14:49:23: ADMIN MSGDEL Requested (DCC Chat)
** 2001-11-26-14:49:33: DCC Send Accepted from DIVX|33!~SLIPPO@IRCGATE-
21470.UNIV-LYON1.FR file: sleepy.rar
** 2001-11-26-14:49:33: Upload Connection Established
** 2001-11-26-14:49:36: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:49:38: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:49:39: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:49:40: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:49:49: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:50:11: Upload: Connection closed: Connection Lost
** 2001-11-26-14:50:12: ADMIN RMUL Requested (DCC Chat)
** 2001-11-26-14:50:13: ADMIN DCLD Requested (DCC Chat)
** 2001-11-26-14:53:30: XDCC LIST queued:
(SYSMA!~SYSMA@194.185.205.IRCGATE-52833)

```

The last log entry is on November 30. Although the intruder returned twice (Dec 5 and Jan 10), it appears those efforts were of little use. On Feb 9, another try was made with some new tools:

```

ls -la dev/pts/01/bnc
total 280
 2 drwxr-xr-x  2 1007    users      1536 Feb  9  2002 ./
 0 drwxr-xr-x  5 root    root        132 Jul 26 03:11 ../
 6 -rw-r--r--  1 1007    users      5307 Dec 31  2000 CHANGES
18 -rw-r--r--  1 1007    users     17982 Dec 31  2000 COPYING
 1 -rw-r--r--  1 1007    users       914 Jul  8  2001 Makefile
12 -rw-r--r--  1 1007    users     12170 Dec 31  2000 README
38 -rwxr-xr-x  1 1007    users     38125 Jul  8  2001 bnc*
 5 -rw-r--r--  1 1007    users      4470 Dec 31  2000 bnc.c
 1 -rw-r--r--  1 1007    users        47 Jul  8  2001 bnc.conf
 1 -rw-r--r--  1 1007    users         6 Jul  8  2001 pid.bnc
[listing truncated]

```

Thanks to the presence of the README file, we can easily tell that this is an IRC “bot” program – an automated agent which connects to an IRC network and listens for

commands. It even has a PID file dated July 8th, indicating that it has probably run each time the system is started since it was installed on the previous November 24th. Note the owning UID is 1007, which does not exist on the system – probably from the original tar file the files were extracted from (assuming they were extracted as root user on the compromised system).

In /dev/pts/01 there is a different set of programs:

```
ls -la dev/pts/01
total 605
 0 drwxr-xr-x  5 root    root      132 Jul 26 03:11 ./
 0 drwxr-xr-x  3 root    root      142 Nov 24 2001 ../
 0 drwxr-xr-x  2 root    root       54 Nov 24 2001 backup/
 2 drwxr-xr-x  2 1007    users    1536 Feb  9 2002 bnc/
 1 -rwxr-xr-x  1 1007    users     82 Feb  9 2002 bnc.conf*
 5 -rwxr-xr-x  1 root    root     4172 Feb  9 2002 cleaner*
 0 drwxr-xr-x  2 root    root      151 Feb  9 2002 etc/
26 -rwxr-xr-x  1 1007    users    26052 Feb  9 2002 identd*
17 -rwxr-xr-x  1 root    root    16772 Feb  9 2002 pg*
556 -rwxr-xr-x  1 1007    users   568336 Feb  9 2002 pico*
 1 -rwxr-xr-x  1 root    bin       12 Feb  9 2002 tmp*
```

'cleaner' is a log file modification script, although its comments are rather ironic in this case:

```
#      Generic log cleaner v0.4 By: Tragedy/Dor (dor@kaapeli.net)
#      Based on sauber..
#
#  This is TOTALLY incomplete... I never added support for IRIX or
#  SunOS...
#  And.. i most likely never will.. And i take no responsibility for
#  any use/misuse
#  of this tool..
#
#  Notes-0.3
#  SunOS support added.. had to rewrite most of it :P
#  Notes-0.4
#  Beta IRIX support added and enabled...
```

So, we get to be the beta-testers for this script! The script itself is extremely simplistic in that it merely removes undesired lines with a grep -v. The 'IRIX support' appears to be a recognition that IRIX usually stores its log files in /var/adm rather than /var/log which is common in Linux. This is not a sophisticated package.

```
ls -la dev/pts/01/etc
total 15
 0 drwxr-xr-x  2 root    root      151 Feb  9 2002 ./
 0 drwxr-xr-x  5 root    root      132 Jul 26 03:11 ../
10 -rw-r--r--  1 root    root    10240 Nov 24 2001 etc.tar
 1 -rwxr-xr-x  1 root    bin       880 Feb  9 2002
ssh_config*
 1 -rw-----  1 root    root      537 Feb  9 2002
ssh_host_key
 1 -rw-r--r--  1 root    root      341 Feb  9 2002
ssh_host_key.pub
 1 -rw-----  1 root    root      512 Jul 26 03:53
ssh_random_seed
```

```

1 -rw-r--r-- 1 root root 4 Jul 11 10:53 sshd.pid
1 -rw-r--r-- 1 root root 541 Feb 9 2002
sshd_config

```

Here we see the config files for the backdoor sshd. The PID file shows it last started on July 11th. Checking sshd_config shows:

```

Port 13000
ListenAddress 0.0.0.0
HostKey /dev/pts/01/etc/ssh_host_key
RandomSeed /dev/pts/01/etc/ssh_random_seed
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes

```

Port 13000 was confirmed as an SSH backdoor by the IDS logs which detected the compromise in the first place, and an external network scan before the system was seized. The backdoor must be started from somewhere at boot time and we'll check that next and try and find the binary. We'll try a brute-force check for the sshd binary by looking for any file with the string sshd_config in it.

```

find . -type f -exec grep -h sshd_config {} \;
./usr/bin/xfsd

```

This is started at the end of the /etc/init.d/xfsd script (which was modified).

Finally, getting back to the /usr/bin/.sh_history file we can tell the intruder is new to this and was having plenty of trouble getting things to work, as can be seen in this segment of the history:

```

ls -l ./usr/bin/.sh_history
3 -rw----- 1 root bin 2296 Jul 26 03:14
./usr/bin/.sh_history
strings ./usr/bin/.sh_history
cd Elite3.1.1
./Copnfg
./Config
./Config
make
ls -la
make
make
make
ls -la
make
/usr/ccs/bin/make
cd ..
rm -rf Elite*
ssh -l root -p 25700 65.101.145.108
exit

```

Obviously, elite-3.1.1 (an IRCd) did not compile properly and the intruder gave up. The timestamp on this file is the last entry into the system before it was taken off the network.

Method of Compromise

Unfortunately, although we can localize the time of compromise to be Nov 26 2001, the exact method is not certain. The system was known to have several vulnerabilities which could have been exploited. One of the most common services attacked during this time was telnetd, which was running on the system at that time. IRIX had multiple remote-root level vulnerabilities with public exploits available for telnetd and anyone with one of the many public exploits could have attained root on the system.

Summary

Given that the compromised system was SGI IRIX, and the lack of forensics tools for dealing with such a system and with the XFS filesystem, the investigation was difficult and tedious. Compounding this was the fact that the user and other parties had access to the system and knowledge of the incident before the seizure of the hardware - during this time attempts were made to "fix" the compromise which only served to cover up the trail even more. In particular, the backdoor sshd binary was irretrievably deleted before the investigator arrived on the scene. Even so, using only traditional tools such as strings, grep, find, and perl, a good understanding of the incident was possible.

It was determined that the intruder had access to the system for a time period of nearly 9 months, although they only appeared to make use of the system intermittently. The intruder attempted several different IRC-related tool installs, of which only one was successfully operated on the system for any length of time. By the presence of DivX tools and DCC copy attempts, it is evident the subject was mainly interested in the system for use as a distribution point for movie files - something it was singularly unsuitable for due to a lack of large amounts of available disk space. The subject clearly did not have much experience with IRIX systems and repeatedly failed to properly compile software for it. Eventually the subject did manage to install an SSH backdoor to the system, which ironically was the action which caused the compromise to be discovered.

© SANS Institute