



# 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>

# Forensic Analysis of a System

## GIAC Certified Forensic Analyst (GCFA) Practical Assignment

Version 1.5  
with Option 1 – Forensic Analysis on a System

Mark Read

29<sup>th</sup> November 2004

© SANS Institute 2005, Author retains full rights.

## Table Of Contents

Part 1 – Analyze an Unknown Image.....	3
Introduction .....	3
Forensic Workstation and Tools .....	3
Analysis Of Image .....	4
Verification of provided image.....	4
Initial Analysis of Image .....	5
Timeline Analysis.....	6
Recovering Deleted Files.....	10
Analysing the contents of a file .....	13
Using Camouflage .....	19
Recovering hidden files .....	24
Conclusion and recommendations .....	28
Legal discussion surrounding the theft of information and use of Steganography .....	30
References.....	32
Part 2 – Forensic Analysis On A System .....	34
Introduction .....	34
Forensic Workstation and Tools .....	36
Incident Response and Seizure of Evidence.....	37
Acquiring the disk image .....	38
Producing a Timeline.....	41
Timeline Analysis .....	43
Log File Analysis .....	49
File System Analysis .....	60
Recovering deleted files .....	65
strings Analysis .....	73
Conclusion and Timeline of Activities .....	76
References.....	78

## Part 1 – Analyze an Unknown Image

### ***Introduction***

For this part of the practical assignment an image of a floppy disk is provided which potentially contains evidence linking the suspect, Robert John Leszczynski, Jr who works for Ballard Industries, with leaking confidential data to a competitor, Rift Inc.

The objective of this part of the assignment is to analyze the floppy disk image using forensically sound methods and tools in order to uncover any evidence linking Robert Leszczynski with leaking the confidential data.

The timeline information already provided is that the floppy disk was seized by security while being taken out of the R&D labs at Ballard Industries on 26<sup>th</sup> April 2004 at approximately 4:45pm MST.

The chain of custody form which is provided is as follows:

- Tag# fl-260404-RJL1
- 3.5 inch TDK floppy disk
- MD5: d7641eb4da871d980adbe4d371eda2ad fl-260404-RJL1.img
- fl-260404-RJL1.img.gz

### ***Forensic Workstation and Tools***

The forensic workstation used to conduct the investigation consisted of the following:

- Compaq Evo N410c
- Linux Operating system – RedHat Enterprise Linux ES Release 3
- VMWare virtual machine running Microsoft Windows XP SP2

The tools used to carry out the investigation were:

- The Sleuthkit (TSK) version 1.72  
Available from <http://www.sleuthkit.org/sleuthkit/>  
Previously known as TASK (The @stake Sleuth Kit)
- Standard Linux base OS commands.

## Analysis Of Image

### Verification of provided image

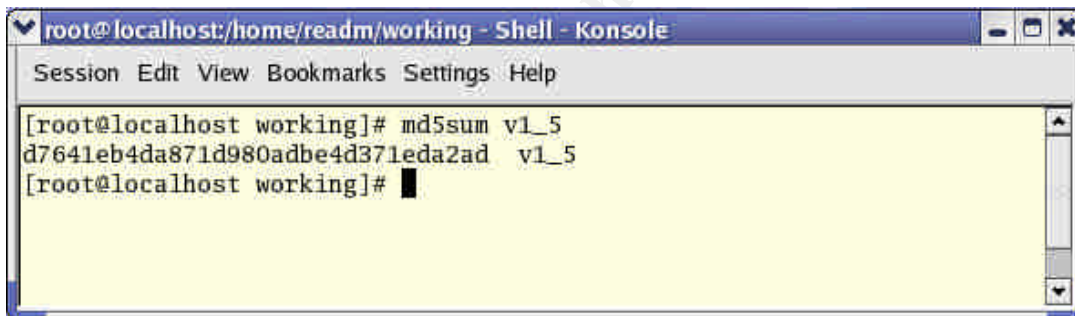
As provided in the assignment directions, the image to be analysed was downloaded from the GIAC website, [http://www.giac.org/gcfa/v1\\_5.gz](http://www.giac.org/gcfa/v1_5.gz). The filename of the downloaded file, *v1\_5.gz*, does differ from the filename detailed on the chain of custody form provided, *fl-260404-RJL1.img.gz*. In the real world this could suggest that the chain of custody is incorrect or not complete. Since this is a theoretical assignment we shall note the slight inconsistency and proceed to verifying that it is actually the same file despite the name.

The file was decompressed using the gunzip utility, and the md5sum command provided the md5 checksum<sup>1</sup> of the file.

The checksum as shown in Figure 1 is the same as detailed on the chain of custody form, thus indicating that it is the same file.

Chain of Custody MD5 checksum:

MD5: **d7641eb4da871d980adbe4d371eda2ad** fl-260404-RJL1.img

A screenshot of a terminal window titled 'root@localhost:/home/readm/working - Shell - Konsole'. The window has a menu bar with 'Session', 'Edit', 'View', 'Bookmarks', 'Settings', and 'Help'. The terminal content shows the command '[root@localhost working]# md5sum v1\_5' followed by the output 'd7641eb4da871d980adbe4d371eda2ad v1\_5'. The prompt '[root@localhost working]#' is shown again on the next line.

```
root@localhost:/home/readm/working - Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@localhost working]# md5sum v1_5
d7641eb4da871d980adbe4d371eda2ad v1_5
[root@localhost working]#
```

Figure 1 - MD5 checksum of the floppy image

<sup>1</sup> MD5 is a one-way hashing algorithm designed to give a unique, consistent and condensed output of a given input.

## Initial Analysis of Image

In order to determine which type of filesystem we are dealing with, the *file* command was used against the image file. The *file* command analyses the header of the given file in order to determine what type of file it is.

```
[root@localhost working]# file v1_5
v1_5: x86 boot sector, system mkdosfs, FAT (12 bit)
```

The image was then mounted so we can browse the filesystem on the forensic workstation. To do this the *mount* command was used. Details of the switches passed are also described.

```
[root@localhost working]# mount -o ro,loop v1_5 /mnt/floppy
```

-o	Specifies there are options to follow
ro	Mount as read-only, so the image cannot be modified.
loop	Use the loop device <sup>2</sup> .
v1_5	Our image file we wish to mount
/mnt/floppy	The mount point where we can access it on our filesystem

Performing a directory listing of our mounted image provides the following list of files. The dates associated with each file details the last modify time.

```
[root@localhost working]# ls -l /mnt/floppy
total 640
-rwxr-xr-x  1 root    root      22528 Apr 23  2004
Acceptable_Encryption_Policy.doc
-rwxr-xr-x  1 root    root      42496 Apr 23  2004
Information_Sensitivity_Policy.doc
-rwxr-xr-x  1 root    root      32256 Apr 22  2004
Internal_Lab_Security_Policy1.doc
-rwxr-xr-x  1 root    root      33423 Apr 22  2004
Internal_Lab_Security_Policy.doc
-rwxr-xr-x  1 root    root      307935 Apr 23  2004
Password_Policy.doc
-rwxr-xr-x  1 root    root      215895 Apr 23  2004
Remote_Access_Policy.doc
```

On the face of it, it looks like the contents of the floppy disk is just policy documents. Opening the files in a standard word processor confirms this. From the details specified in the original timeline, the floppy disk was seized from Mr Leszczynski on 26<sup>th</sup> April which indicates these files were possibly last written approx 3-4 days prior.

Documents such as these also contain metadata that provide details of the original author, company, and in some instances also record who has viewed or modified the document. Upon examination of the metadata within these

---

<sup>2</sup> The loop device allows you to mount an image of a file system stored as a standard file as though it was an external disk.

documents nothing of great significance was found. All documents with the exception of two were attributed to Ballard Industries, Inc. as being the authoring company. Password\_Policy.doc and Remote\_Access\_Policy.doc were attributed to Cisco Systems, Inc.

While much of the metadata can be viewed by simply viewing *Properties* within the application itself, further information can be obtained by using a hex editor - which we'll see in action later. In this instance though, no further information was available.

## Timeline Analysis

The next stage is to generate a timeline so we can see some specifics on when the files on disk were actually created, as well as last modified and accessed. To do this involves a two stage process.

Firstly, we will use the *fls* tool to compile data about the files on the floppy image, and then secondly use the *mactime* tool, to take the output of *fls* and put it into a readable timeline format. Both the *fls* and *mactime* tools are part of The Sleuthkit

So, first, let's run the *fls* tool. Details of the switches passed are described below.

```
[root@localhost working] fls -f fat12 -m / -r v1_5 > timeline.fls
```

-f fat12	Specifies the filesystem. This was previously determined using <i>file</i> command <sup>3</sup> .
-m /	Output in mactime format and put a / before every name
-r	Recurse directories
v1_5	The image we wish to analyse
> timeline.fls	The output file storing the results

The timeline.fls file produced contains details of every file, including deleted ones (subject to there still being an entry in the FAT table for it of course), on the image. We specify the *-m* switch so that it outputs in a format that we can then read into the mactime tool.

```
0|/Internal_Lab_Security_Policy1.doc (INTERN~1.DOC) |0|13|33279|/-/  
rwxrwxrwx|1|0|0|0|32256|1082908800|1082622666|1082943982|512|0
```

I've included a single line from the timeline.fls file generated as a result of running the command. As you can see, the format isn't the easiest to read, unless of course you've can convert Epoch time format (the number of seconds since January 1<sup>st</sup> 1970) to real life time format in your head.

---

<sup>3</sup> To determine the exact parameter to pass, the fls command was run with the --help parameter and a list of available file systems and the appropriate parameters to pass were listed.

To convert it into a more readable format, the mactime tool is used. This will also separate the modify, access and create times into separate lines and sort into time and date order for you as well.

```
[root@localhost working]# mactime -b timeline.flc > timeline.mac
```

-b	Specifies the location of the “body” file to read in, which contains the raw data.
timeline.flc	The body file
> timeline.mac	The output file that will store the results

Once run, we then end up with the timeline.mac file which is our timeline in a more readable format.

© SANS Institute 2005, Author retains full rights.



**Figure 2 – Timeline of the floppy image**

Sat Feb 03 2001 19:44:16	36864	m..	-/-rwxrwxrwx	0	0	5	/CamShell.dll (_AMSHLL.DLL) (deleted)
Thu Apr 22 2004 16:31:06	33423	m..	-/-rwxrwxrwx	0	0	17	/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
	32256	m..	-/-rwxrwxrwx	0	0	13	/Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
Fri Apr 23 2004 10:53:56	727	m..	-/-rwxrwxrwx	0	0	28	/_ndex.htm (deleted)
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	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)
	215895	.a.	-/-rwxrwxrwx	0	0	23	/Remote_Access_Policy.doc (REMOTE~1.DOC)
	32256	.a.	-/-rwxrwxrwx	0	0	13	/Internal_Lab_Security_Policy1.doc (INTERN~1.DOC)
	307935	.a.	-/-rwxrwxrwx	0	0	20	/Password_Policy.doc (PASSWO~1.DOC)
	36864	.a.	-/-rwxrwxrwx	0	0	5	/CamShell.dll (_AMSHLL.DLL) (deleted)
	33423	.a.	-/-rwxrwxrwx	0	0	17	/Internal_Lab_Security_Policy.doc (INTERN~2.DOC)
	727	.a.	-/-rwxrwxrwx	0	0	28	/_ndex.htm (deleted)
Mon Apr 26 2004 09:46:18	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)

The timeline is formatted as follows.

First column contains the date and time that refers to the change, whether it be modify, access, create, or any combination of all three.

The second column shows the size of the file on disk.

The third column details as to what the entry relates to.

m	Modified
a	Access
c	Create

Modify means the last time the file was written to.

Access refers to the time at which the file itself was last read in some form. This may just be the header, not necessarily the entire file. This time being updated is subject to the media being writable though. If the floppy disk is write protected, then this is not updated.

Create relates to last time the directory entry was updated. A create time could be updated as a result of a file being renamed for example, as well as its original creation of course.

The fourth column displays the permissions of the file. This is split in to three sections, owner; group; and the world. Each section contains a placeholder for three settings, (r)ead, (w)rite, and e(x)ecute. As an example, a file which has all permissions for the owner, only read permissions for the group, and only execute permissions for everyone else would look like `rw-r--r--`.

The fifth column details the owner, and sixth details the group.

Seventh column provides the directory entry or inode number.

And the final column is the filename itself.

Since the image is of a FAT12 filesystem the permissions and user/group information is not applicable as they are not supported by this format.

What immediately jumps out at you is that the last access time for all files is dead on midnight on Monday April 26<sup>th</sup>. This could be quite legitimate, but does not feel quite right. Also the modify time for all files is before the create time. That's also a little strange. A possible explanation for this is that the floppy disk was used in two different computers where the clock on each differs quite significantly. It could also be that all of the files were renamed. Another explanation is that the MAC times have been tampered with, so information on the timeline should not be taken as gospel. The volume label was also changed to R JL, which coincidentally are also the initials of Robert John Leszczynski. If there is something on this disk that is incriminating and in the letters do relate to your initials initials, would you really be changing the label of the disk so that people can trace it back to you?

Apart from the policy documents which we previously viewed, there are two additional files to include that have since been deleted, `CamShell.dll` and `_ndex.htm`.

## Recovering Deleted Files

Being that so far we've not yet actually found anything interesting within the live content on the disk image, let's see if we can recover the two deleted files.

Looking at the timeline in figure 2, the column immediately preceding the filename is the directory entry number, or inode. The directory entry contains information about the file, but more importantly it contains information on where the file is physically on the disk. In the case of `_ndex.htm` this number is 28.

```
Mon Apr 26 2004 09:47:36 727 ..c -/-rwxrwxrwx 0 0 28 /_ndex.htm (deleted)
```

Before we go any further, for our own information let's take a look at some more details about our two deleted files. We will use the TSK *istat* tool in order to view all details regarding a directory entry. Let's start with `_ndex.htm`.

```
[root@localhost working]# istat -f fat12 v1_5 28
```

-f fat12	Specifies the image is a fat12 filesystem
v1_5	Our image
28	The directory entry number

```
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
```

From the information above, we can see that the file occupies sectors 33 and 34. To actually recover the file we'll use the TSK *icat* tool, which will work this information out for itself. We just need to provide the directory entry number.

```
[root@localhost working]# icat -r -f fat12 v1_5 28 > deleted/_ndex.htm
```

-r	Specifies that the file is to be recovered. Without this, only the first sector would be outputted.
-f fat12	Specifies that the image is a fat12 filesystem
v1_5	The floppy image
28	The directory entry number

deleted/_index.htm	The output file, where the contents of the recovered file will be stored.
--------------------	---

The deleted/\_index.htm is then viewed, which gives us...

```
<HTML>
<HEAD>
<meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
<TITLE>Ballard</TITLE>
</HEAD>
<BODY bgcolor="#EDED"ED">

<center>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"

codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflas
h.cab#version=6,0,0,0"
  WIDTH="800" HEIGHT="600" id="ballard" ALIGN="">
  <PARAM NAME=movie VALUE="ballard.swf"> <PARAM NAME=quality VALUE=high>
<PARAM
NAME=bgcolor VALUE=#CCCCC> <EMBED src="ballard.swf" quality=high
bgcolor=#CCCCC WIDTH="800" HEIGHT="600" NAME="ballard" ALIGN=""
  TYPE="application/x-shockwave-flash"
  PLUGINSPage="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</center>
</BODY>
</HTML>
```

A pretty standard HTML file. Nothing special.

Just in case we need to recover this file again and prove that it is the same one, let's perform a MD5 hash of it.

```
[root@localhost deleted]# md5sum _ndex.htm
17282ea308940c530a86d07215473c79 _ndex.htm
```

Let's now look at the other file, CamShell.dll. From the timeline information we can see that it is directory entry number 5. Here's what *istat* tells us about it.

```
[root@localhost working]# istat -f fat12 v1_5 5
Directory Entry: 5
Not Allocated
File Attributes: File, Archive
Size: 36864
Num of links: 0
Name: _AMSHLL.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:

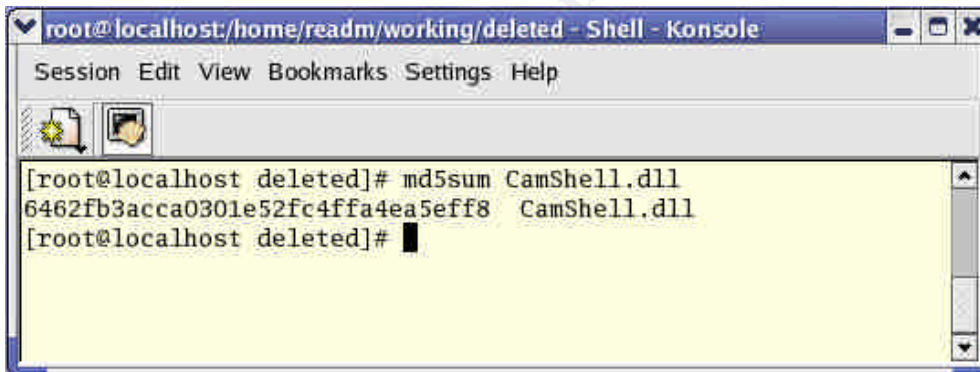
```
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104
```

First two sectors are 33 and 34, the same as `_index.htm`. Since the modify time of `_index.htm` is later than `CamShell.dll` then there is a very high probability (subject to the mac times being correct) that the first two sectors of `CamShell.dll` have been overwritten by `_index.htm`. Given that we've also successfully recovered `_index.htm` then it is pretty much a certainty.

We've essentially lost those two sectors. Not the end of the world though as there may be useful information within the remaining content that we can use to proceed further.

```
[root@localhost working]# icat -r -f fat12 v1_5 5 > deleted/CamShell.dll
```

And let's not forget to perform a MD5 hash of the partially recovered file.



```
[root@localhost deleted]# md5sum CamShell.dll
6462fb3acca0301e52fc4ffa4ea5eff8 Camshell.dll
```

## Analysing the contents of a file

*strings* is a standard Linux command that allows for the extraction of consecutive ASCII characters from a binary file. By default, 4 or more consecutive ASCII characters will be returned. Essentially the purpose of this tool is to return content that could potentially be a word that may mean something when read by a person.

So, let's run strings against the file we have just recovered.

```
[root@localhost deleted]# strings CamShell.dll
```

Unfortunately in this instance no ground-breaking information was returned. As suspected, the HTML code from the recovered `_index.htm` file was included that took up the first two sectors. Other than that there was nothing really my more that gives any more clues, except the following line.

```
C:\My Documents\VB Programs\Camouflage\Shell\IctxMenu.tlb
```

The name of the package that this dll was part of (if indeed it is a dll, as we're purely going by the filename at this point) is possibly called Camouflage and is a Visual Basic app. Essentially though, other than a potential name there was nothing more of any value. Searching the Internet for the term "camouflage" as suspected returned a lot of unrelated results.

The next step is to use a hex editor and see if we can spot anything else that could give us something more to go on.

After scrolling through the contents manually, some interesting information showed its head. Starting at address location 0000:7240 (HEX), we see the following.

```

0000:7230 39 00 30 00 34 00 42 00 30 00 00 00 64 00 4c 00 9.0.4.B.0...d.L.
0000:7240 01 00 43 00 6f 00 6d 00 6d 00 65 00 6e 00 74 00 ..C.o.m.m.e.n.t.
0000:7250 73 00 00 00 68 00 74 00 74 00 70 00 3a 00 2f 00 s...h.t.t.p.:/.
0000:7260 2f 00 77 00 77 00 77 00 2e 00 63 00 61 00 6d 00 /.w.w.w...c.a.m.
0000:7270 6f 00 75 00 66 00 6c 00 61 00 67 00 65 00 2e 00 o.u.f.f.l.a.g.e..
0000:7280 66 00 72 00 65 00 65 00 73 00 65 00 72 00 76 00 f.r.e.e.s.e.r.v.
0000:7290 65 00 2e 00 63 00 6f 00 2e 00 75 00 6b 00 00 00 e...c.o...u.k...
0000:72a0 54 00 32 00 01 00 43 00 6f 00 6d 00 70 00 61 00 T.2...C.o.m.p.a.
0000:72b0 6e 00 79 00 4e 00 61 00 6d 00 65 00 00 00 00 00 n.y.N.a.m.e.....
0000:72c0 54 00 77 00 69 00 73 00 74 00 65 00 64 00 20 00 T.w.i.s.t.e.d. .
0000:72d0 50 00 65 00 61 00 72 00 20 00 50 00 72 00 6f 00 P.e.a.r. .P.r.o.
0000:72e0 64 00 75 00 63 00 74 00 69 00 6f 00 6e 00 73 00 d.u.c.t.i.o.n.s.
0000:72f0 00 00 00 00 b0 00 88 00 01 00 46 00 69 00 6c 00 ....0....F.i.l.
0000:7300 65 00 44 00 65 00 73 00 63 00 72 00 69 00 70 00 e.D.e.s.c.r.i.p.
0000:7310 74 00 69 00 6f 00 6e 00 00 00 00 00 4b 00 65 00 t.i.o.n.....K.e.
0000:7320 65 00 70 00 73 00 20 00 66 00 69 00 6c 00 65 00 e.p.s. .f.i.l.e.
0000:7330 73 00 20 00 63 00 6f 00 6e 00 74 00 61 00 69 00 s. .c.o.n.t.a.i.
0000:7340 6e 00 69 00 6e 00 67 00 20 00 73 00 65 00 6e 00 n.i.n.g. .s.e.n.
0000:7350 73 00 69 00 74 00 69 00 76 00 65 00 20 00 69 00 s.i.t.i.v.e. .i.
0000:7360 6e 00 66 00 6f 00 72 00 6d 00 61 00 74 00 69 00 n.f.o.r.m.a.t.i.
0000:7370 6f 00 6e 00 20 00 73 00 61 00 66 00 65 00 20 00 o.n. .s.a.f.e. .
0000:7380 66 00 72 00 6f 00 6d 00 20 00 70 00 72 00 79 00 f.r.o.m. .p.r.y.
0000:7390 69 00 6e 00 67 00 20 00 65 00 79 00 65 00 73 00 i.n.g. .e.y.e.s.
0000:73a0 2e 00 00 00 cc 00 a8 00 01 00 4c 00 65 00 67 00 ....0....L.e.g.
0000:73b0 61 00 6c 00 43 00 6f 00 70 00 79 00 72 00 69 00 a.l.C.o.p.y.r.i.
0000:73c0 67 00 68 00 74 00 00 00 43 00 6f 00 70 00 79 00 g.h.t...C.o.p.y.

```

Figure 3 - Screenshot from a hex editor displaying the contents of recovered file CamShell.dll.

From this we can procure the following information.

Comments	<a href="http://www.camouflage.freeseve.co.uk">http://www.camouflage.freeseve.co.uk</a>
CompanyName	Twisted Pear Productions
FileDescription	Keeps files containing sensitive information safe from prying eyes
ProductName	Camouflage

First step then is to visit the advertised website to see if we can get more information on this. Unfortunately that appears to be a dead link as that subdomain did not exist.

Performing an Internet search using Google<sup>4</sup> for the terms “twisted pear productions” and “camouflage”, however, did returned some results.

The website at <http://camouflage.unfiction.com/> entitled “Camouflage Home Page – Hide your files!” looks like it may be the answer we’re looking for.

*“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.”*

– Taken from the Overview page on the Camouflage website

<sup>4</sup> Google is a popular Internet search engine. <http://www.google.com>



We're not 100% though that this is the same site as was originally hosted at the address found within the file. A search for <http://www.camouflage.freemove.co.uk> at the Wayback Machine<sup>5</sup> allows us to see the site again in all its glory. Viewing an archived copy from March 14<sup>th</sup> 2003 the page looks exactly the same as the current live at <http://camouflage.unfiction.com/>.

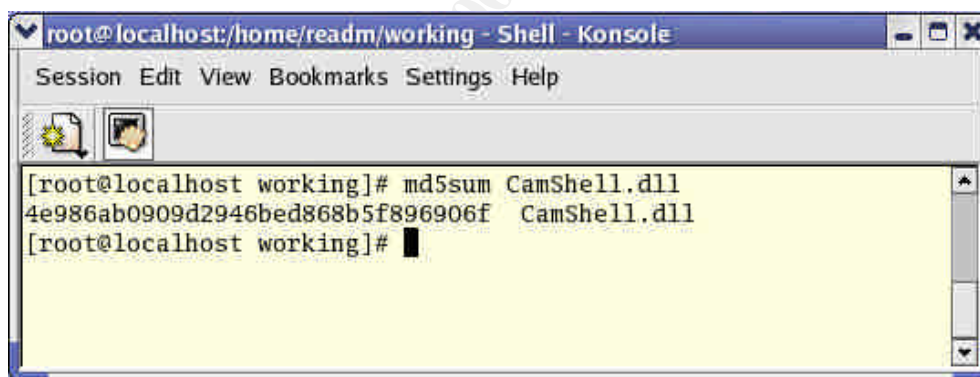
The Camouflage program uses a process called Steganography which is the science of hiding information within information.

A paper written by Joshua Silman entitled *Steganography and Steganalysis: An Overview*<sup>6</sup> gives a very good introduction to this process and the techniques used to detect its existence. Examples such as invisible ink and microdots were classic examples where information was hidden using this method so as to not give away its existence.

In the computer age this technique is still alive and well, but instead information is being electronically hidden, the most common method being the hiding of whatever you want to hide within standard image files.

Before we proceed any further though we need to ensure that the package here contains the CamShell.dll file that we have found and it is in fact the same file. We shall therefore download and install the file.

Once installed a file called CamShell.dll is found in the c:\program files\Camouflage directory. This file is copied over to our analysis workstation and a MD5 hash is created of it.

A screenshot of a terminal window titled 'root@localhost:/home/readm/working - Shell - Konsole'. The window has a menu bar with 'Session', 'Edit', 'View', 'Bookmarks', 'Settings', and 'Help'. Below the menu bar is a toolbar with icons for a file, a folder, and a search. The terminal text shows a command prompt '[root@localhost working]# md5sum CamShell.dll' followed by the output '4e986ab0909d2946bed868b5f896906f CamShell.dll'. The prompt is then followed by another '#' character.

```
[root@localhost working]# md5sum CamShell.dll
4e986ab0909d2946bed868b5f896906f CamShell.dll
[root@localhost working]#
```

```
[root@localhost working]# md5sum CamShell.dll
4e986ab0909d2946bed868b5f896906f CamShell.dll
```

At this stage to confirm that the deleted file we recovered and the one that was just installed as part of the Camouflage package are the same we would simply

<sup>5</sup> The Way Back Machine (<http://www.waybackmachine.org>) is an archive of practically every website visitable on the internet with snapshots taken over time. With it you can view the contents of an archived website from any point in its history.

<sup>6</sup> Steganography and Steganalysis: An Overview; Joshua Silman;  
<http://www.sans.org/rr/whitepapers/steganography/553.php>



perform a MD5 hash of the recovered file and then compare the two. Unfortunately since the first two sectors of the recovered CamShell.dll file were overwritten by \_ndex.htm this is not possible. We will have to take another approach.

Opening the recovered and new CamShell.dll files side by side in hex editors, a quick glance through gives the impression that the two files are the same. For the time being, therefore, that is what we shall assume.

This being the case, if we were to take a copy of the first two sectors from the new CamShell.dll file and then pasted them over the first two sectors of the recovered CamShell.dll file, then in theory we should end up with two files exactly the same. And to prove this we can then perform the MD5 hash check.

Firstly, in order to find out exactly how much we need to copy from the beginning of the new CamShell.dll file we need to know what 2 sectors actually equates to.

To provide more detailed information on the filesystem, the TSK *fsstat* tool is used against the image. The *-f* switch is used to specify the filesystem type, fat12.

```
[root@localhost working]# fsstat -f fat12 v1_5
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

METADATA INFORMATION
-----
Range: 2 - 45426
Root Directory: 2

CONTENT INFORMATION
-----
Sector Size: 512
Cluster Size: 512
```

Total Cluster Range: 2 - 2840

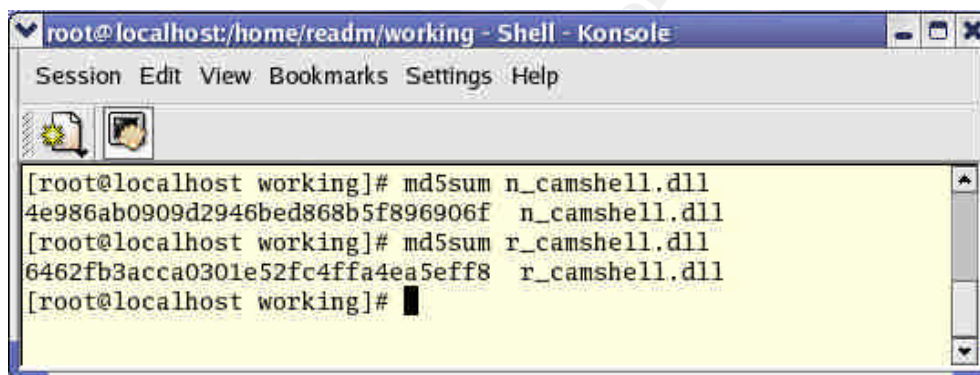
FAT CONTENTS (in sectors)

```
-----  
105-187 (83) -> EOF  
188-250 (63) -> EOF  
251-316 (66) -> EOF  
317-918 (602) -> EOF  
919-1340 (422) -> EOF  
1341-1384 (44) -> EOF
```

From this we can see that the sector size of the disk image is 512 bytes. As we want the first two sectors that means we need the first 1024 bytes.

Since we are now dealing with two files of the same name, to avoid confusion the copy we recovered from the floppy image will be renamed as `r_camshell.dll`, and the copy of the new file that was installed as part of the Camouflage package will be renamed as `n_camshell.dll`. This in no way affects our original image.

So that we are clear where we're starting from, a MD5 checksum is performed on both files.



```
root@localhost:/home/readm/working - Shell - Konsole  
Session Edit View Bookmarks Settings Help  
[root@localhost working]# md5sum n_camshell.dll  
4e986ab0909d2946bed868b5f896906f  n_camshell.dll  
[root@localhost working]# md5sum r_camshell.dll  
6462fb3acca0301e52fc4ffa4ea5eff8  r_camshell.dll  
[root@localhost working]#
```

```
[root@localhost working]# md5sum n_camshell.dll  
4e986ab0909d2946bed868b5f896906f  n_camshell.dll  
[root@localhost working]# md5sum r_camshell.dll  
6462fb3acca0301e52fc4ffa4ea5eff8  r_camshell.dll
```

The two files are then opened up in khxedit, a hex editor. The first 1024 bytes from `n_camshell.dll` is then copied into the clipboard.

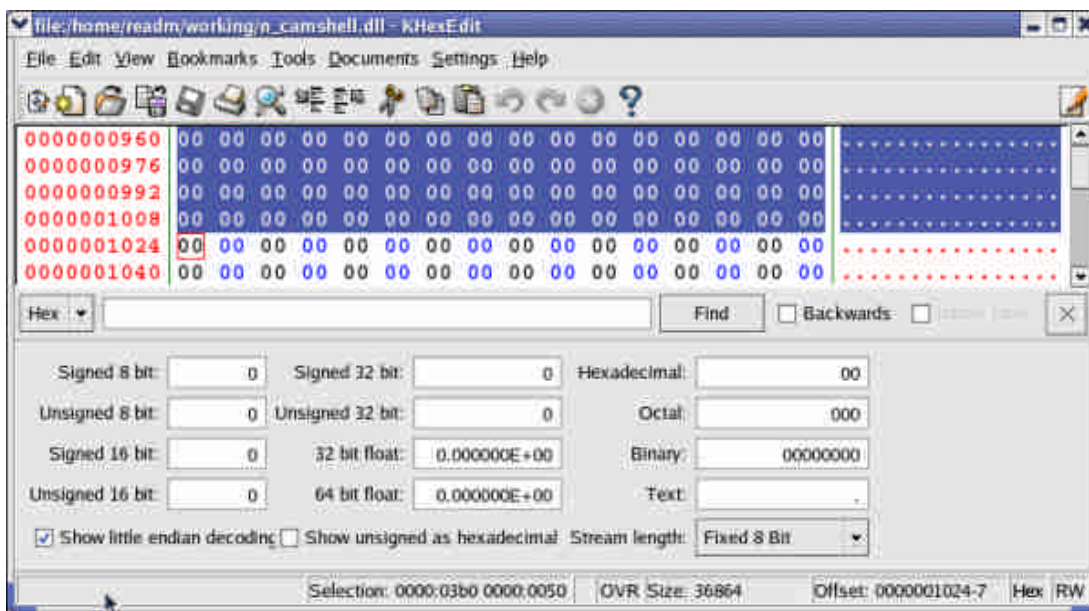
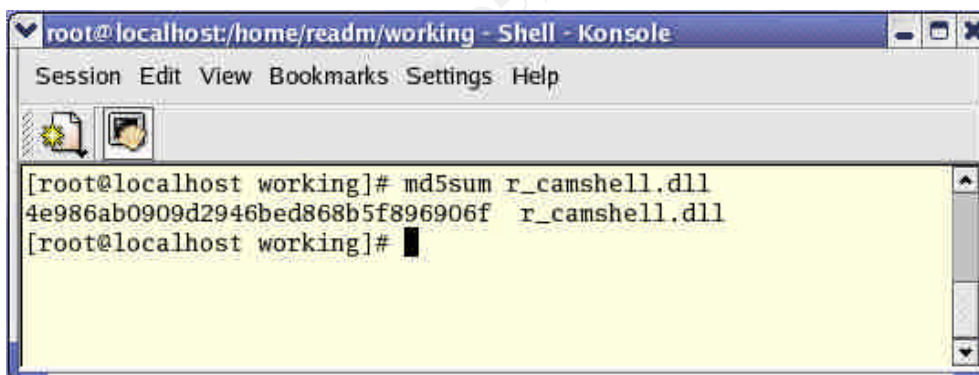


Figure 4 - Selection of first 1024 bytes in khedit

The first 1024 bytes of the `r_camshell.dll` file is then deleted and the contents of the clipboard is pasted in its place.

The `r_camshell.dll` file is then saved and a MD5 checksum is performed on the new file.



```
[root@localhost working]# md5sum r_camshell.dll
4e986ab0909d2946bed868b5f896906f r_camshell.dll
[root@localhost working]#
```

The MD5 checksum of the recovered file (`r_camshell.dll`) now matches that of the one that was installed as part of the Camouflage package (`n_camshell.dll`).

We now have confirmation that the original recovered file belongs to the Camouflage package that we downloaded.

There is a fairly high chance that the innocent looking policy documents on the disk may contain slightly more than previously thought.

## Using Camouflage

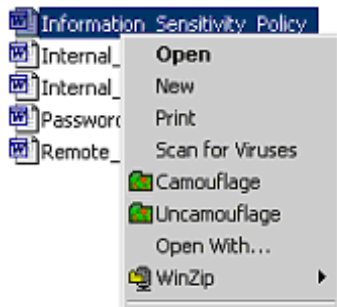


Figure 5 - Right-Click menu in Explorer

Once Camouflage is installed, an option is added to the right-click menu within Windows Explorer on a Microsoft Windows system that permits for you to Camouflage or Uncamouflage a file.

To perform the required action, simply select a file, right-click the mouse, and then select the appropriate option.

To demonstrate how Camouflage works, we are going to take a jpg image file, a plain text file, and camouflage one inside the other. Since one file will be hidden inside another, it is advisable that a binary file is used as the one that will be doing the hiding, otherwise you'll end up with your "hidden" file appearing as gobbledygook when you open it up in something like notepad.



Figure 6 - image1.jpg

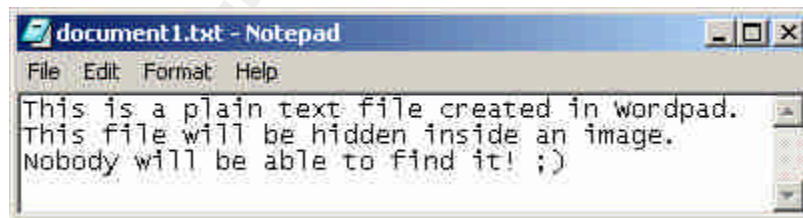
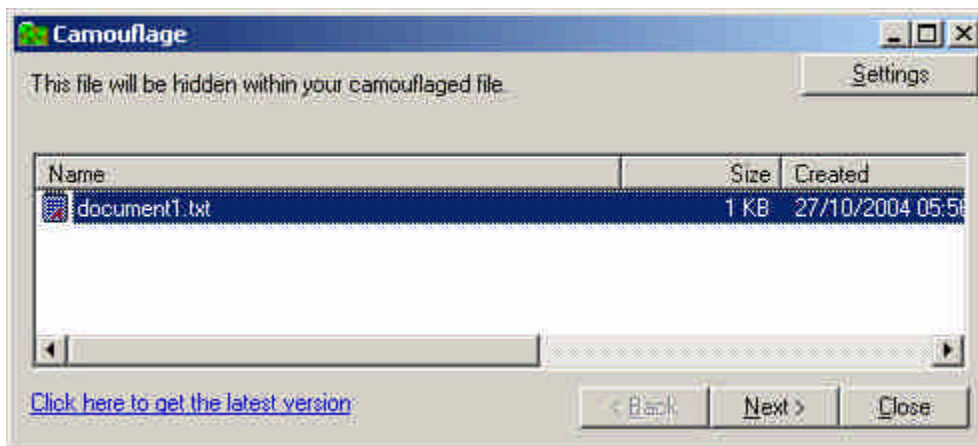


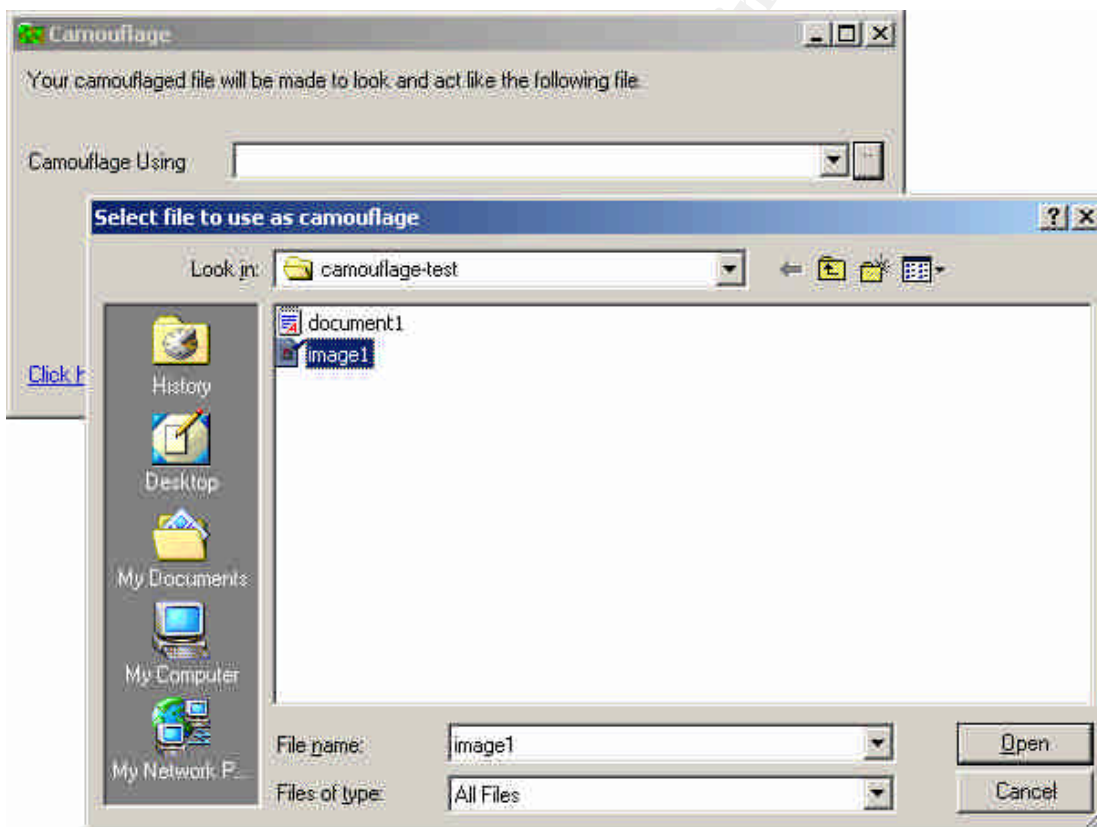
Figure 7 - document1.txt

The first step is to select document1.txt, which is the file we wish to hide within image1.jpg and select Camouflage from the right-click menu.

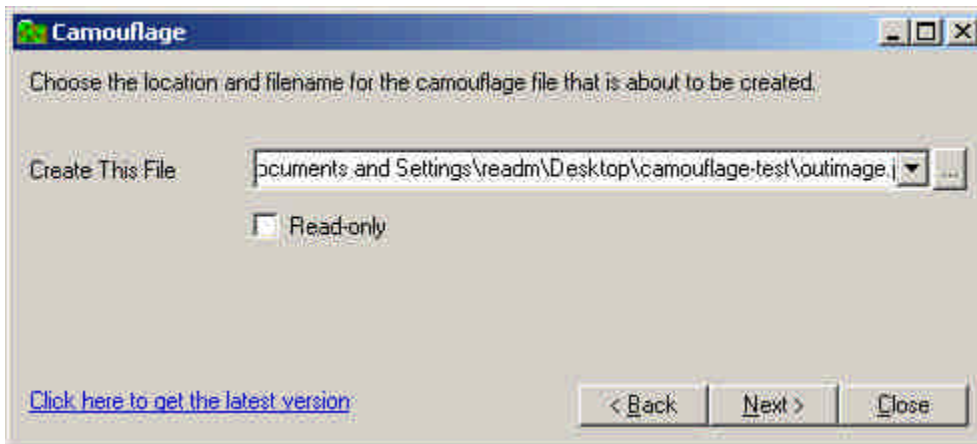
A window appears asking for confirmation that the file we have selected is the one we wish to hide.



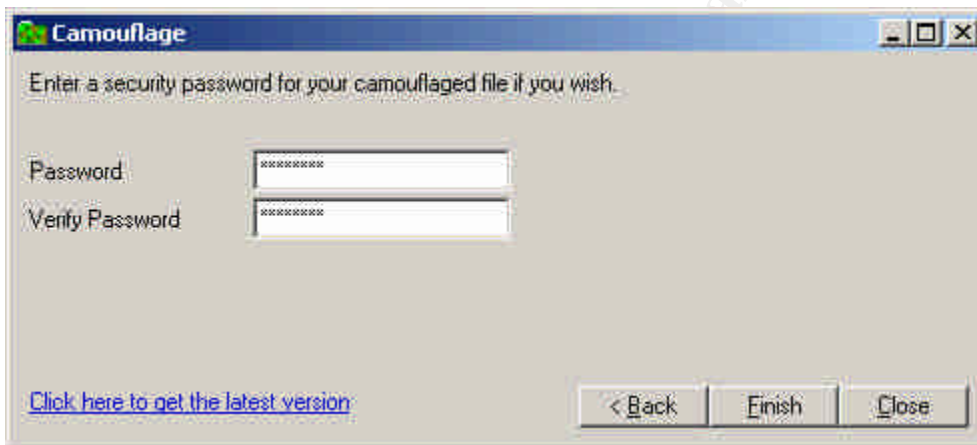
After selecting the obligatory Next > button, we're then asked which file we'd like to use as camouflage. In our case, we select image1.jpg.



The next stage is to specify the output file, which is the the file which will hide our secret message. We will call it outimage.jpg



The final stage is to specify a password. One does not have to be given, but since we're going to the trouble of trying to hiding something it sort of makes sense to do so.



And we're done.

If we now open up outimage.jpg in a image viewer we'll see that it looks exactly the same as the original image1.jpg. No sign of the hidden document!



Figure 8 - outimage.jpg

Whilst the images themselves look no different, if we compare the file sizes of the two images, we'll see that outimage.jpg is approximately 1k larger than image1.jpg. This is because of course it now also has the contents of document1.txt inside it as well.



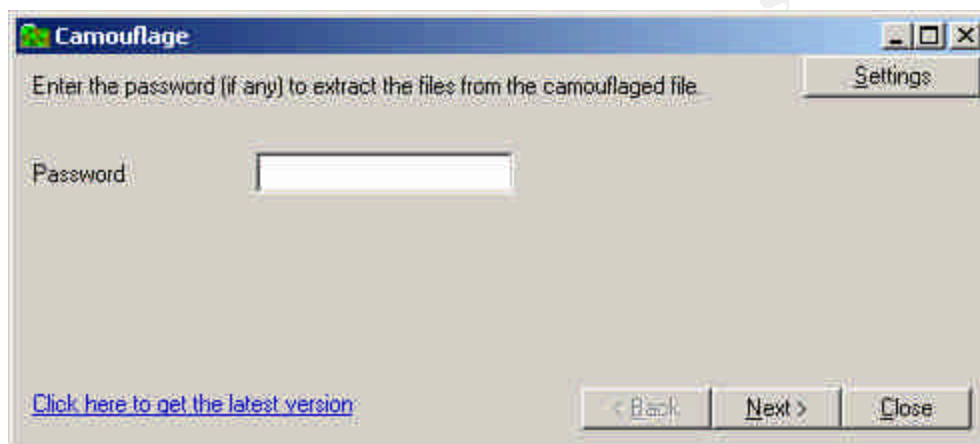
image1	3 KB	JPEG Image
document1	1 KB	TXT File
outimage	4 KB	JPEG Image

**Figure 9 - File sizes of original and camouflaged files**

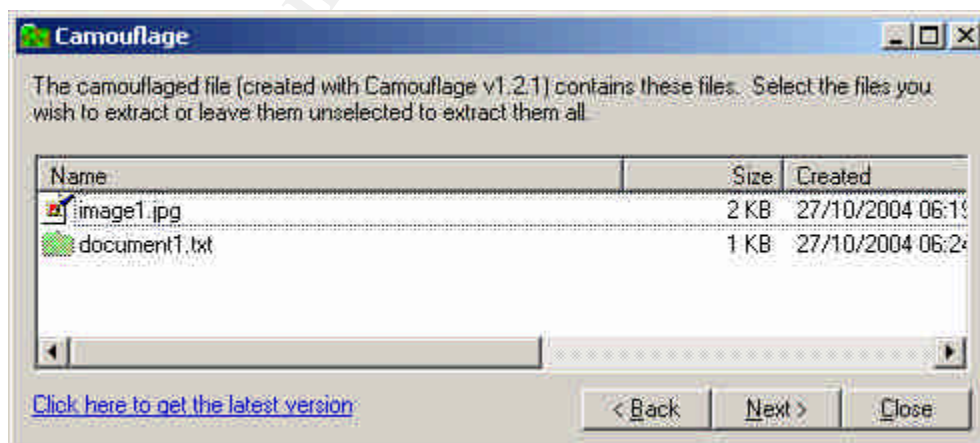
Now we have a text document hidden within an image, let's extract it so that we can read our secret message.

We right click this time on the generated outimage.jpg file and select Uncamouflage from the menu. This then presents us with a request for a password.

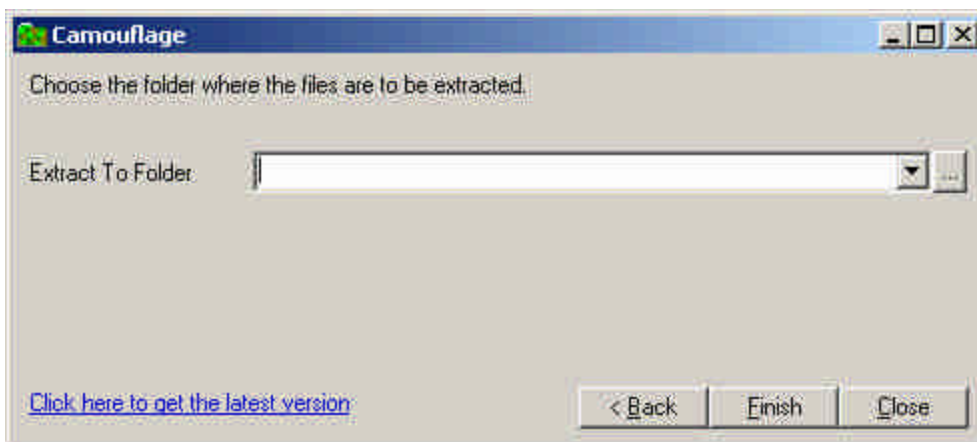
This does not necessarily mean that the file has some hidden content in it. This prompt will appear if you attempt to uncamouflage any file.



Upon entering the correct password, details of the hidden contents of the file are shown. In this instance we can see that the original image1.jpg is there along with the file we hid, document1.txt.



Clicking *Next >* then prompts for a location to save the two files.



We give a location, click Finish and the job is done. We've now successfully extracted the original image1.jpg and the hidden text document, document1.txt.

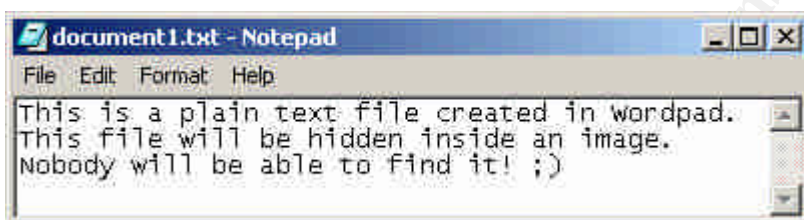


Figure 10 - The extracted hidden file, document1.txt

Job done! We can now see our secret message.

What exactly does it do to the file though? In order to see what is happening under the hood, let's view the original image1.jpg and the generated outimage.jpg files in a hex editor.

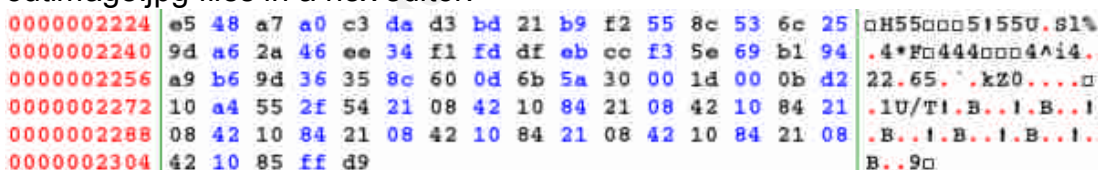


Figure 11 - image1.jpg

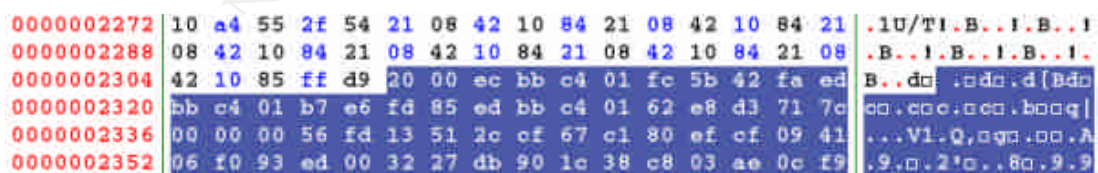


Figure 12 - outimage.jpg

Up until location 2308 both files are exactly the same. At this address, the image1.jpg file ends, but the outimage.jpg still contains data right through until location 3287. The content of this addition data is unreadable and contains no "English" words. We can safely assume that this is our document1.txt, but has been encrypted or scrambled in some way.



The same applies to other file types tested. The hidden file is appended in all cases using the Camouflage application. Depending upon the file type, and whether there is any defined structure that is consistent at the end of the file, it may be possible to tell whether any additional data has been hidden by viewing it in this way. In instances where you could not normally detect the end of a file, then it may not be as easy.

## Recovering hidden files

The documents from the floppy image were transferred to our Windows XP VMWare session and the uncamouflage process was run against them. No password was specified when prompted.

This resulted in only one success. The file Internal\_Lab\_Security\_Policy.doc had an additional file, Opportunity.txt, camouflaged within it.

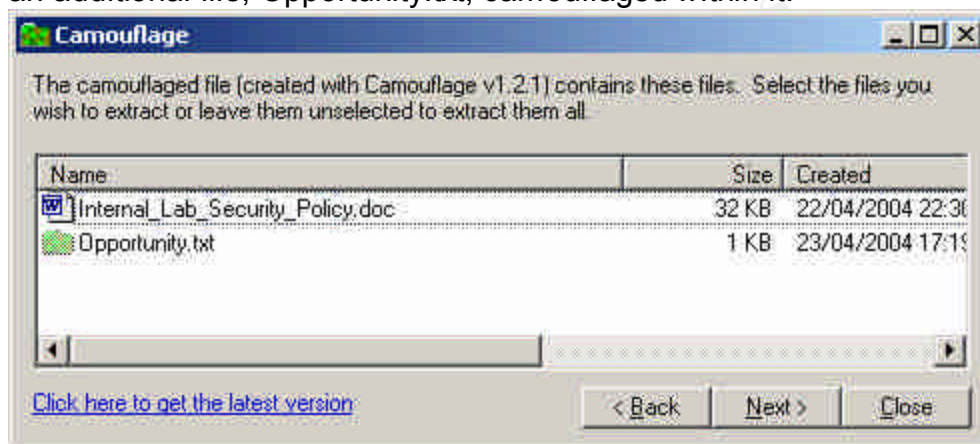
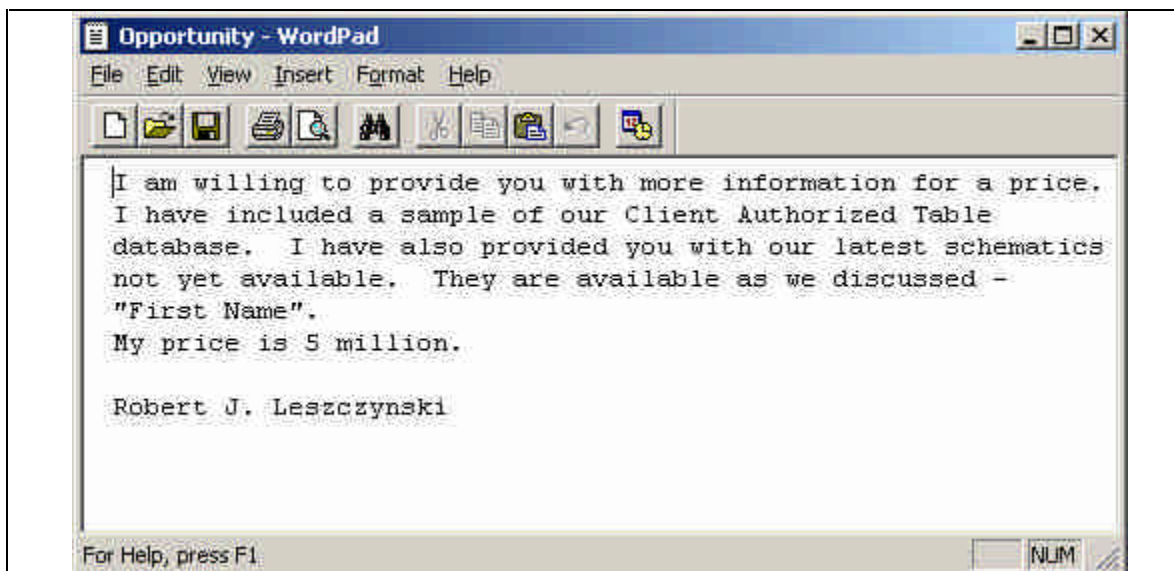


Figure 13 - Camouflaged files within Internal\_Lab\_Security\_Policy.doc

The Opportunity.txt file was extracted and viewed. The results can be seen below.

<b>Original File</b>	Internal_Lab_Security_Policy.doc
<b>Camouflage Password:</b>	<none>
<b>Extracted Files:</b>	<b>MD5 Checksum</b>
Internal_Lab_Security_Policy.doc	e0c43ef38884662f5f27d93098e1c607
Opportunity.txt	3ebd8382a19c88c1d276645035e97ce9



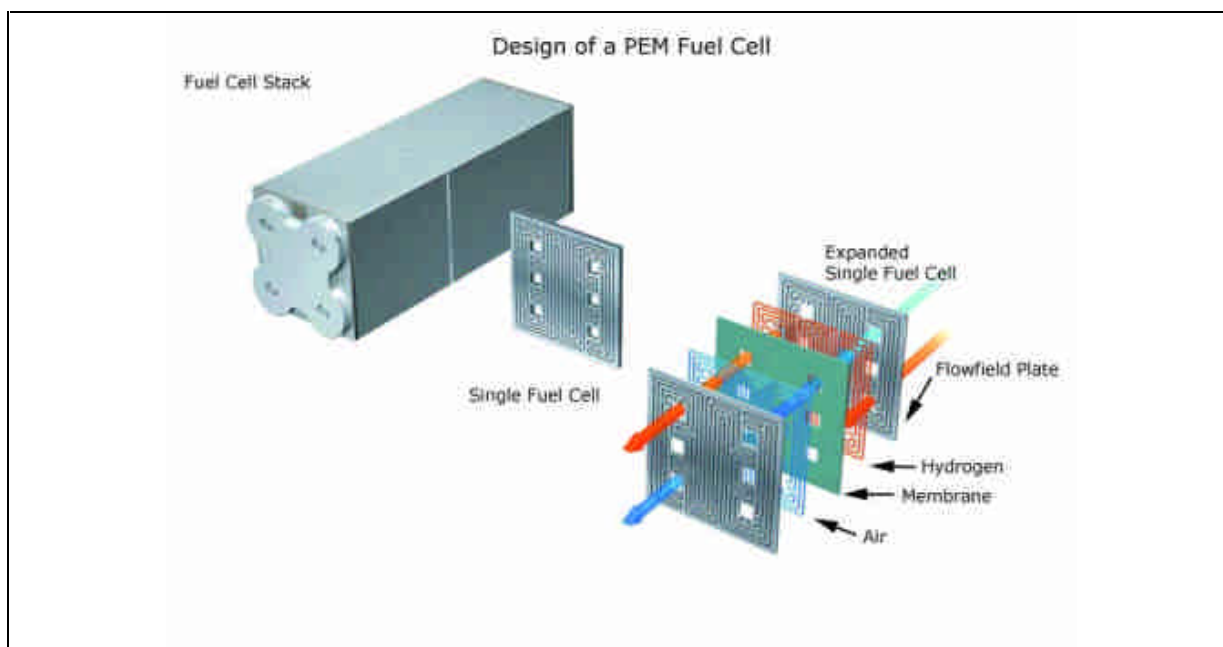
The contents of the file appears to be a note from Robert J. Leszczynski detailing that he is willing to provide more information for a price and that he has also provided the person to which this note is addressed a sample of a Client Authorized Table database and latest schematics which are not available. The contents of the note suggest that the database and schematics may also be camouflaged somewhere on the disk. The 'They are available as we discussed – "First Name"' comment could be a hint as to the password used for additional camouflaged files.

Following this lead, attempts to uncamouflage the other files on the disk using a series of anticipated passwords was performed.

Firstly, it was suspected that "First Name" may be Mr Leszczynski's first name, Robert. This password provided no results.

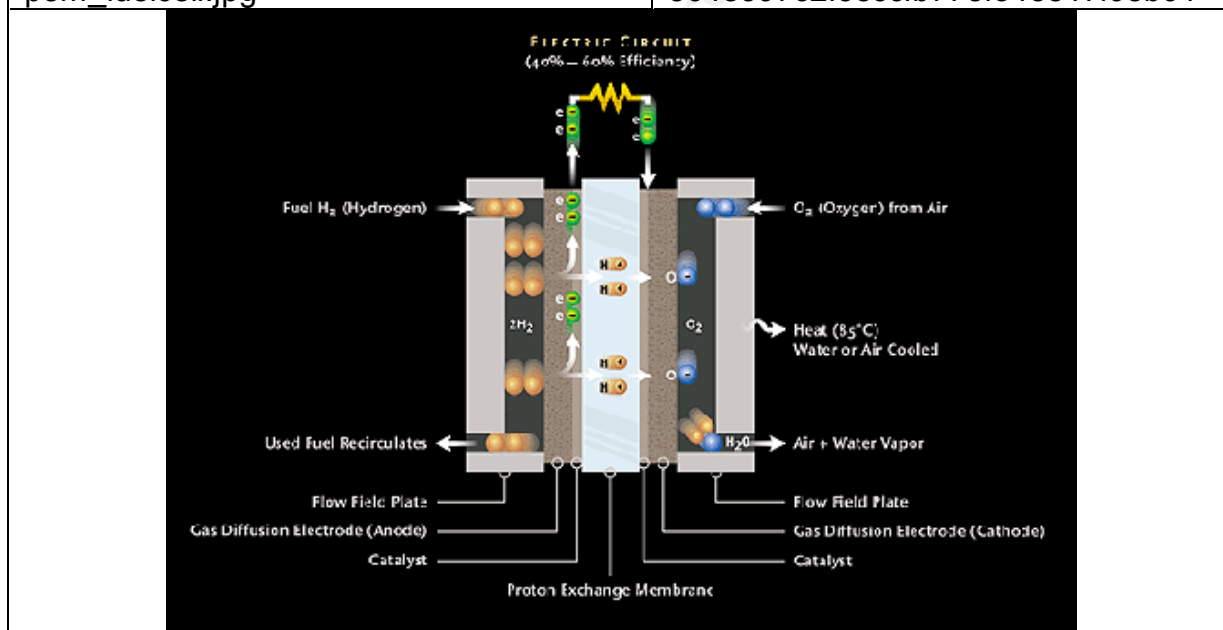
It was then noted that the filenames used for the documents on the disk consisted of more than one word, or name. Passwords which consisted of the first word of the document file name were used, which did produce results from two of the files.

<b>Original File</b>	Password_Policy.doc
<b>Camouflage Password:</b>	Password
<b>Extracted Files:</b>	<b>MD5 Checksum</b>
Password_Policy.doc	e5066b0fb7b91add563a400f042766e4
PEM-fuel-cell-large.jpg	5e39dcc44acccdca7bba0c15c6901c43



pem\_fuelcell.jpg

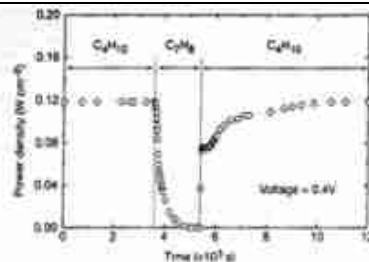
864e397c2f38ccfb778f348817f98b91



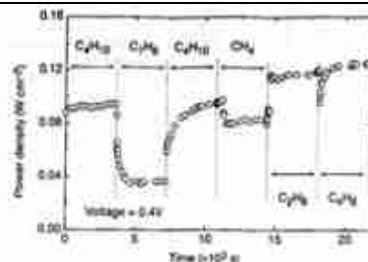
Hydrocarbon%20fuel%20cell%20page2.jpg

9da5d4c42fdf7a979ef5f09d33c0a444





**Figure 3** Effect of switching fuel type on the cell with the Cu-ceria composite anode at 973 K. The power density of the cell is shown as a function of time. The fuel was switched from *n*-butane ( $C_4H_{10}$ ) to toluene ( $C_7H_8$ ) and back to *n*-butane.



**Figure 4** Effect of switching fuel type on the cell with the Cu-edged ceria composite anode at 973 K. The power density is shown as a function of time. The fuels were: *n*-butane ( $C_4H_{10}$ ), toluene ( $C_7H_8$ ), *n*-butane, methane ( $CH_4$ ), ethane ( $C_2H_6$ ), and 1-butene ( $C_4H_8$ ).

higher temperature. Visual inspection of a cell after two days in *n*-butane at 1,073 K showed that the anode itself remained free of the tar deposits that covered the alumina walls.

Although it is possible that the power generated from *n*-butane fuels resulted from oxidation of  $H_2$ —formed by gas-phase reactions of *n*-butane that produce hydrocarbons with a lower C:H ratio—other evidence shows that this is not the case. First, experiments were conducted in which the cell was charged with *n*-butane and

the current density returned to  $0.12 \text{ W cm}^{-2}$  after one bout. Because the return was not instantaneous, it appears that carbon formation occurred during exposure to toluene, but that the anode is self-cleaning. We note that the electrochemical oxidation of soot has been reported by others<sup>11</sup>.

The data in Fig. 4 show that further improvements in cell performance can be achieved. For these experiments, yttria-doped ceria was substituted for ceria in the anode, and the current

Original File

Camouflage Password:

Extracted Files:

Remote\_Access\_Policy.doc

CAT.mdb

Remote\_Access\_Policy.doc

Remote

MD5 Checksum

2afb005271a93d44b6a8489dc4635c1c

869ff6b71c7be3eb06b6635c864b1

First	Last	Phone	Company	Address	Address2	City	State	Zipcode	Account	Password
Bob	Eurotech	703-233-3346	Cosk Latic	345 Main St		Alexandria	VA	22301	mspmann	yP43HMH
Jerry	Jackson	410-677-7223	Double Jy	11553 W 27 St		Baltimore	MD	21270	jack27st	4JW3Pg5
David	Lee	666-554-0322	Tech Vision	300 Lane Grove Lane		Wichita	KS	30789	leedwche	DTA2643
Mary	Horton	800-254-4mg	King Labs, Inc	700 King Lake Ave	Suite 300	Blinn	MS	39533	horking	YK754pA
Larry	Jones	877-Get-down	Quick Printing	55 E. Grand View Dr		Omaha	NE	68108	joneset	882y4884
Jeff	Hayes	404-895-5521	Big Sky First	90 Old Saw Mill Rd		Bilings	MT	50330	hayrocks	3R306b7
Roger	Foresteel	710-886-2312	TGFS	100 Greenfield Rd		Austin	TX	77238	foreste	sd0W8Uv
Edward	Cash	212-662-9997	E & C Int.	75 S. King St	Suite 300	Santa Barbara	CA	90124	cashemg	QWu21K
Steve	Ewi	616-833-0125	Ayland Labs	66 New Way		Honolulu	HA	96991	ewi@mw	3CH20105
Jodie	Kelly		Data Movers	2295 Bayview Ave	Suite 110	Wethersby	U.K.	LE22 8R3	kellys	tmu08r3c
Patrick	Rey		The Magic Lam	4150 Regents Park	Row #170	Calgary	CAN	R4T16DE	mythema	Uag6000

## ***Conclusion and recommendations***

Information which could prove to be very useful to a competitor was found on the floppy disk that Mr Leszczynski was attempting to take off of the premises. Together with that information a note was also found detailing that further information was available at a price, leaving no doubt that this information was being sold. Furthermore, it is obvious that attempts were made to conceal this information by the use of Steganography.

No conclusive proof could be found that this information had been provided to Rift Inc., nor that it was Mr Leszczynski that provided it. There are a number of unanswered questions that could possibly cast doubt on the guilt of Mr Leszczynski.

The floppy disk was seized from Mr Leszczynski's briefcase after the information had already been leaked. The timeline, if it is to be believed, shows that the information uncovered was written to disk several days prior. If this is the case, then why has Mr Leszczynski left it to now to attempt to remove the disk from the premises?

From the original report we already know that this information has already been leaked, which suggests that the contents of the disk have already been removed from the premises once in order to hand it over to the competitor. That being the case, why would Mr Leszczynski risk bringing it back onto the premises? More importantly, why would he take the further risk of sneaking the disk back out again without concealing any new information on it. This is of course assuming the evidence found on the disk is the same information that is believed to have been leaked to the competitor – this therefore needs to be verified. There is of course also the possibility that the Mr Leszczynski is planning to provide this information to another company as well and this was not actually destined for Rift Inc.

As the disk contains incriminating evidence, why would Mr Leszczynski change the volume name on the disk to his initials?

Based on this I would say there are enough questions to start believing that there is a possibility that it was not Mr Leszczynski who leaked the information. This, however, is only a hunch and nothing should be decided without hard evidence.

Moving forward, in order to hopefully obtain more conclusive evidence I would suggest that any PCs used by Mr Leszczynski are examined to see if the Camouflage program is, or was ever installed, and if the operating system supports it determine whether it was installed using Mr Leszczynski's user account.

If there are any audit logs of who has accessed the policy documents that were found on the floppy disk from their original stored location, whether it be on an internal website or from a file server, then these should be examined to possibly highlight any other potential suspects.

If any audit logs of Internet activity are kept, these should be examined to see if anybody has visited the Camouflage website in the past.

If CCTV exists in the Lab it should be examined at around the times indicated by the timeline that the files were written to see if there is any footage of anybody using and then concealing a floppy disk, or generally looking suspicious.

© SANS Institute 2005, Author retains full rights.

## ***Legal discussion surrounding the theft of information and use of Steganography***

The evidence uncovered was attempted to be concealed using a method called Steganography. This does raise the question as to whether any crime was actually committed by concealing stolen information in this way, and if so what the legal consequences are under UK law.

Let's first look at what crime has been committed. Confidential company information was removed from the premises. While this in itself is not breaking any laws, since information is not a physical item and as such is therefore not classed as theft in the traditional sense, most if not all companies will have policies in place to ensure company confidential data remains safe, and acknowledgment of these policies will be written into employees contracts. A Computer Weekly article written by Bill Goodwin in March 2003<sup>7</sup> highlights this exact issue and how the police are powerless to take any action due to the outdated computer crime laws. As quoted by Detective Sergeant Santorelli, a senior investigator at The Computer Crime Unit, *"We are talking about a huge amount of money being made by people stealing networked data. It is just an anachronism that stealing data is not illegal. The bottom line is that if someone phones up and says my database has been stolen, I can only refer them to a civil remedy."*

The only action companies can take is to pursue the civil action route; the employee is dismissed from their job for gross misconduct and a claim for damages is made against them.

The Computer Misuse Act 1990<sup>8</sup> does provide laws against the unauthorised access to data. Deliberately and knowingly accessing data that you are not permitted to is classed as an offence under section 1 of the act which carries a punishment of no more than 6 months in prison, a fine not exceeding level 5 (currently £5,000 at time of writing<sup>9</sup>) on the standard scale, or both. If it can be proved that the person who stole the data was not authorised to access that data in the first instance, then a criminal offence would have been committed.

Does the fact that Steganography was used provide a legal case at all though? The answer is simple, no. There is no evidence under UK law that the use of Steganography itself breaks any laws. The fact that it was used to conceal the information being stolen though does enhance the case and can rule out any claims that the data was accidentally left on the disk, etc. Using a case of shoplifting as an example, a person leaving a store with the stolen article in their hand could claim that they quite legitimately forgot they were carrying it and depending upon other evidence could talk their way out of it and may not be charged. If the person left with the article tucked inside their coat or concealed using any other method then they could not claim such innocence. Under the eyes of the law an offence was committed in both instances, but one would be seen more favourable than the other.

---

<sup>7</sup> <http://www.computerweekly.com/articles/article.asp?liArticleID=119870&liFlav>

<sup>8</sup> [http://www.hmso.gov.uk/acts/acts1990/Ukpga\\_19900018\\_en\\_1.htm](http://www.hmso.gov.uk/acts/acts1990/Ukpga_19900018_en_1.htm)

<sup>9</sup> [http://www.cjsonline.gov.uk/offender/community\\_sentencing/fine/](http://www.cjsonline.gov.uk/offender/community_sentencing/fine/)

The use of Steganography does lead however to an interesting slant on how the Regulation Of Investigatory Powers Act 2000<sup>10</sup> would be viewed in this instance. In instances where encrypted material is discovered as part of an investigation, under the RIPA Act authorities are permitted to insist that the key is provided in order to unlock the encryption and provide the unencrypted data. Failure to provide the key to the “protected data” could result in a prison sentence not exceeding two years, a fine or both upon conviction.

Any data hidden by use of Steganography would also be classed as “protected data” under this act, and as the key to unlock the data must be provided upon request. The problem is that encryption and Steganography differ in quite a significant way.

The purpose of encrypting data is to ensure it is secure. It is similar to data being placed within a locked safe. The safe does not need to be hidden in order to keep the data secure.

The purpose of using Steganography is to ensure that data remains hidden. It is similar to data being hidden under the floorboards or under a mattress. Hiding it in these locations does not make the data secure, it purely aids in it not being discovered in the first place.

In the instance of the safe, you are required to provide the key to unlock it. In the instance of hiding under the floorboards, it has to be found before any request to unlock it is given.

What happens in instances where the authorities have proof that Steganography has been used but cannot find or detect the files being used as the “camouflage”? If the Steganography method used to hide the data is good and cannot be detected using known methods of Steganalysis, then a request to provide the key to unlock it cannot be made.

Understandably, the use of Steganography is becoming increasingly popular with criminals wishing to hide their crimes or evidence of crimes, and has become a major concern for the authorities. An example of which is highlighted in an article written by Declan McCullagh for Wired entitled “Bin Laden: Steganography Master?”<sup>11</sup> In it he describes how terrorists are using Steganography to hide maps and photographs of targets.

---

<sup>10</sup> <http://www.legislation.hmso.gov.uk/acts/acts2000/20000023.htm>

<sup>11</sup> “Bin Laden: Steganography Master?” - <http://www.wired.com/news/print/0,1294,41658,00.html>



## References

"Camouflage Home Page - Hide your files!".

URL: <http://camouflage.unfiction.com/Overview.html> (22<sup>nd</sup> November 2004)

Silman, Joshua. "Steganography and Steganalysis: An Overview". 1.2f. August 2001. URL: <http://www.sans.org/rr/whitepapers/steganography/553.php> (22<sup>nd</sup> November 2004)

Goodwin, Bill. "Police hamstrung by UK's outdated computer laws".

ComputerWeekly.com 6<sup>th</sup> March 2003. URL:

<http://www.computerweekly.com/articles/article.asp?liArticleID=119870&liFlav> (22<sup>nd</sup> November 2004)

McCullagh, Declan. "Bin Laden: Steganography Master?". Wired News 7<sup>th</sup>

February 2001. URL: <http://www.wired.com/news/print/0,1294,41658,00.html> (22<sup>nd</sup> November 2004)

Her Majesty's Stationary Office . "Computer Misuse Act 1990". 29<sup>th</sup> June 1990.

URL: [http://www.hmsso.gov.uk/acts/acts1990/Ukpga\\_19900018\\_en\\_1.htm](http://www.hmsso.gov.uk/acts/acts1990/Ukpga_19900018_en_1.htm) (22<sup>nd</sup> November 2004)

Criminal Justice System. "Fines". Community Sentencing.

URL: [http://www.cjsonline.gov.uk/offender/community\\_sentencing/fine/](http://www.cjsonline.gov.uk/offender/community_sentencing/fine/) (22<sup>nd</sup> November 2004)

Her Majesty's Stationary Office. "Regulation of Investigatory Powers Act 2000 Chapter 23" .Regulation of Investigatory Powers Act 2000. 28<sup>th</sup> July 2000.

URL: <http://www.legislation.hmsso.gov.uk/acts/acts2000/20000023.htm> (22<sup>nd</sup> November 2004)

Google. "Google" URL: <http://www.google.com> (22<sup>nd</sup> November 2004)

Internet Archive. "Wayback Machine" URL: <http://www.waybackmachine.org>

(22<sup>nd</sup> November 2004).

RedHat, Inc. "RedHat Linux". URL: <http://www.redhat.com> (22<sup>nd</sup> November 2004)

Hewlett Packard. "Compaq". URL: <http://www.compaq.com> (22<sup>nd</sup> November 2004)

Carrier, Brain. "The Sleuthkit". 1.72. September 7<sup>th</sup> 2004.

URL: <http://www.sleuthkit.org/sleuthkit/> ( 22<sup>nd</sup> November 2004)

@stake, Inc. "The @stake Sleuth Kit". URL: <http://www.atstake.com> (22<sup>nd</sup> November 2004)

Global Information Assurance Certification. "GIAC". URL: <http://www.giac.org> (22<sup>nd</sup> November 2004)

VMWare, Inc. "VMWare" URL: <http://www.vmware.com> (22<sup>nd</sup> November 2004)

Microsoft Corporation. "Microsoft Windows". URL: <http://www.microsoft.com> (22<sup>nd</sup> November 2004)

© SANS Institute 2005, Author retains full rights

## Part 2 – Forensic Analysis On A System

### Option 1

#### ***Introduction***

The second part of the practical assignment is to perform a forensic analysis of a live system that has been compromised and is in an unknown state to the investigator.

In order to obtain such a system to investigate, a honeypot<sup>12</sup> was created. Due to lack of hardware, the honeypot itself was installed within a VMWare<sup>13</sup> virtual machine on a PC running Microsoft Windows XP SP2 as the host operating system.

The operating system used for the honeypot was a RedHat Linux 6.2 build with “everything<sup>14</sup>” installed.

On Thursday November 4<sup>th</sup> 2004 the unpatched honeypot was placed on an isolated network connected to the internet via an ADSL line. Since network address translation was taking place on the ADSL router, configuration changes were made to ensure that all traffic for all ports were routed to the IP address of the honeypot.

The default Apache test page was presented to anybody connecting to the machine via HTTP, which in itself is usually an open invitation to be hacked as it suggests the box probably hasn't been configured as well as it should.

An script was written to automatically check the default page served by the Apache server to detect any changes, and would alert in the instance of it being defaced. A network sniffer was also put in place to monitor for outbound SYN packets which would highlight any traffic originating from the server. In this instance there should be little or no outbound traffic originating from the server, so monitoring for this activity should give a clear sign should somebody have compromised the box and is now attempting to communicate with the outside world.

For 9 days no unusual events were observed, a number of ident request were seen to originate from the honeypot which is a normal for when an smtp session is established. On Saturday November 13<sup>th</sup> at 18:51 GMT an active FTP session was observed with the data channel originating from the server. At 21:50 GMT a change was detected on the default web page. At 22:30 GMT the change was confirmed as a defacement, as shown below.

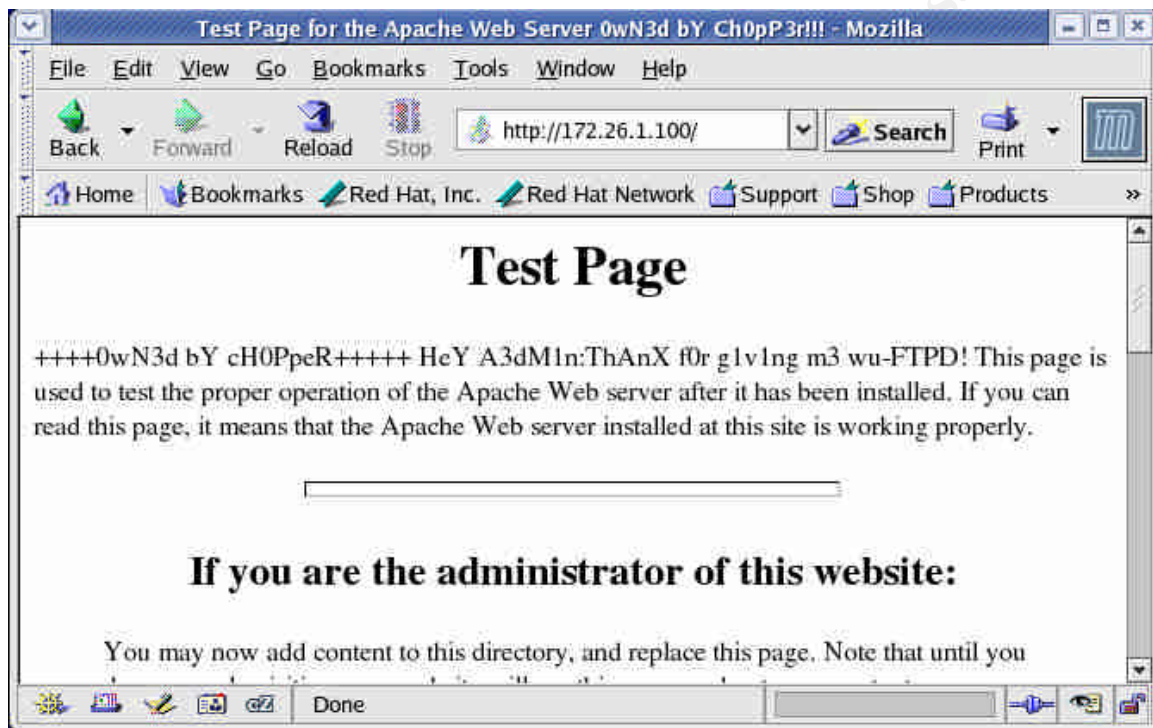
---

<sup>12</sup> A honey pot is a sacrificial computer system that is designed to be compromised by hackers in order to study their methodologies.

<sup>13</sup> VMWare is a product originally made by VMWare Inc (<http://www.vmware.com>) that allows for a virtual operating system to be run while the primary operating system is available for use. VMWare virtual machines are completely segregated from the host, although a path between the two may be established through configuration options within the software.

<sup>14</sup> During installation the user is given a choice of what packages should be installed. An alternative option that is given is to install absolutely everything.

Since the server had no active logged in console or legitimate remote telnet sessions, a conscious decision was made to pull the mains power and treat this as a dead system. Logging into the server in order to capture volatile data such as active network connections and live memory could risk evidence being destroyed, either by the normal login process touching files which would therefore change the MAC<sup>15</sup> times, or by the possibility of something being added by the attacker to the login script of active accounts that could cause damage to the system upon a successful login.



**Please note that the IP addresses detailed in this report relating to those used by the hacker have been modified so as to protect the guilty. To ensure consistency and accuracy of any investigation based upon IP address, the substitute addresses are within the same original subnet and therefore relate to the same service provider.**

<sup>15</sup> MAC - Modify, Access and Create times. A record is kept about the last time these activities took place for every file on the system.

## ***Forensic Workstation and Tools***

The forensic workstation used to conduct the instigation consisted of the following:

- Compaq Evo N410c
- Linux Operating system – RedHat Enterprise Linux ES Release 3

The tools used to carry out the investigation were:

- The Sleuthkit (TSK) version 1.72  
Available from <http://www.sleuthkit.org/sleuthkit/>  
Previously known as TASK (The @stake Sleuth Kit)
- Knoppix STD (Security Toolkit Distribution) v0.1b  
Knoppix is a bootable cd contain a linux distribution allowing you to perform many tasks without requiring a hard disk.  
The original Knoppix distribution is available from <http://www.knoppix.org>  
Knoppix STD is available from <http://www.knoppix-std.org>
- Standard Linux base OS commands.

## ***Incident Response and Seizure of Evidence***

Incident Number: fi1001  
Date: 13<sup>th</sup> November 2004  
Time: 22:30 GMT  
Location: Home premises of Mark Read

### **Items seized:**

fi1001-01	Non-branded PC Tower
fi1001-02	Fujitsu hard disk drive Model: MPF3204AH Serial: NO.01149072

### **VMWare image details:**

Location:	d:\RedHat6.2\RedHat6.2.vmx						
Size:	4GB						
SHA1 Hash of drive image:	Unavailable						
Partitions:	<table><tr><td>sda1</td><td>Linux Partition SHA1 Hash: 5e9d839848ca63f42788fcac5e53d7d2b7f9a7b3</td></tr><tr><td>sda2</td><td>Extended Partition SHA1 Hash: 70cc8b5c0282e5114fd3f3688571d9623bdf7214</td></tr><tr><td>sda5</td><td>Swap Partition SHA1 Hash not available</td></tr></table>	sda1	Linux Partition SHA1 Hash: 5e9d839848ca63f42788fcac5e53d7d2b7f9a7b3	sda2	Extended Partition SHA1 Hash: 70cc8b5c0282e5114fd3f3688571d9623bdf7214	sda5	Swap Partition SHA1 Hash not available
sda1	Linux Partition SHA1 Hash: 5e9d839848ca63f42788fcac5e53d7d2b7f9a7b3						
sda2	Extended Partition SHA1 Hash: 70cc8b5c0282e5114fd3f3688571d9623bdf7214						
sda5	Swap Partition SHA1 Hash not available						

### **Chain of custody:**

13 <sup>th</sup> November 2004 22:40 GMT	Mark Read Acquisition of vmware image. Seizure of Items.
---	--

### **Case background:**

Microsoft Windows XP SP2 primary operating system running RedHat Linux 6.2 virtual server using VMWare software.  
RedHat Linux 6.2 server web server home page defaced on 13<sup>th</sup> November 2004 at approximately 21:50 GMT

### **Action taken:**

PC powered off by removing mains cable. Image taken of VMWare virtual disk on site. PC seized, bagged and tagged.

## ***Acquiring the disk image***

As no equipment was available to provide a disk to disk copy of the evidence the PC was powered back on. As it was the virtual server within the VMWare session that was compromised, booting the host operating system of Microsoft Windows XP was deemed safe.

The compromised guest VMWare operating system would have to be booted to enabled access to the virtual drive in order to obtain an image copy. So as not to alter anything on the disk image, the VMWare session was booted from a Knoppix STD CDROM. During the boot sequence the bios configuration was accessed to ensure that the CDROM was the primary boot device before the hard disk itself.

To ensure that the integrity of the disk can be left without any doubt, a hash of the disk image needed to be established.

The preferred method of doing this is using the md5sum tool which will produce a unique MD5 hash value attributed to the disk image. Unfortunately it was discovered that the version of the md5sum tool distributed with the Knoppix STD cdrom was flawed.

A known bug exists with old versions of md5sum whereby it cannot cope with performing a hash of a file larger than 2GB. The resulting output upon attempting is a report of "Success" instead of the hash itself.

To save messing around, the sha1sum tool which also existed on the Knoppix STD cdrom was chosen to produce the hash instead. sha1sum is similar to md5sum, except it uses the sha1 algorithm instead of md5. While sha1 is believed to be better, md5 is traditionally still used as it is considered the standard.

Using sha1sum, a sha1 hash was produced of the sda drive. The drive image was then copied across the network to the forensic workstation using the dd tool. Once the copy was complete sha1sum was run against the copy and the two hashes were compared. They did not match.

There have been no known reports that booting from and acquiring disk images using Knoppix touches the physical disk, which is one explanation of why the two checksums did not match, so this had to be investigated further.

sha1sum was run a further two times against /dev/sda and in both instances the sha1 hash that was produced differed. It was now important to discover what was changing.

Using the fdisk command, details of the drive geometry was obtained.

```
root@0[forensics]# fdisk -l /dev/sda
Device      Boot  Start  End  Blocks  Id  System
/dev/sda1   *      1     505   4056381  83  Linux
/dev/sda2             506   522   136552+  5   Extended
/dev/sda5             506   522   136521   82   Linux Swap
```

There were 3 partitions listed with the Linux Swap being part of the Extended partition as the start and end blocks of sda2 and sda5 are the same. sha1sum was run against each partition in turn to obtain a unique sha1 hash. When it was run against /dev/sda5 an error was produced. Running md5sum also produced an error.

```
root@0[forensics]# sha1sum /dev/sda5
sha1sum: /dev/sda5: Input/output error

root@0[forensics]# md5sum /dev/sda5
error processing /dev/sda5: failed in buffer_read(fd): mdfile:
Input/output error
```

sha1sum was run again against all partitions and it was found that the hash for sda1 and sda2 had not changed, but again an error was produced when run against sda5.

It is safe to say that the contents of sda1 and sda2 had not changed, and it is the error that is being produced by sda5 that is causing the inconsistency. No explanation can be given for the strange results on this that are being produced from this partition.

To be sure, sha1sum was run once more, and the resulting hashes recorded.

```
root@0[forensics]# sha1sum /dev/sda1
5e9d839848ca63f42788fcac5e53d7d2b7f9a7b3  /dev/sda1

root@0[forensics]# sha1sum /dev/sda2
70cc8b5c0282e5114fd3f3688571d9623bdf7214  /dev/sda2
```

Using the dd tool, copies of sda1, sda2 and sda5 were made across the network to the forensic workstation. To do this, the following commands were run.

On the Knoppix booted VMWare session:

```
root@0[forensics]# dd if=/dev/sda1|nc 172.26.1.103 2001 -w 3 &
root@0[forensics]# dd if=/dev/sda2|nc 172.26.1.103 2002 -w 3 &
root@0[forensics]# dd if=/dev/sda5|nc 172.26.1.103 2005 -w 3 &
```

On the forensic workstation:

```
[root@localhost fi1001]# nc -l -p 2001 > fi1001.sda1.dd &
[root@localhost fi1001]# nc -l -p 2002 > fi1001.sda2.dd &
[root@localhost fi1001]# nc -l -p 2005 > fi1001.sda5.dd &
```

dd is a tool used to make bit copies (exact) of files, disks and partitions. As the copy needs to end up on a the forensic workstation for analysis, it is used in combination with the netcat tool, or nc for short.

Netcat is often referred to as the Swiss Army knife of tools as it is very powerful and offers many options. In this instance the output of dd was piped to nc in order to transfer data across the network to the forensic workstation.

On the forensic workstation three instances of netcat (nc) were set up listening on tcp ports 2001, 2002 and 2003. Whatever was received on each of these



ports was then copied out into individual files; fi1001.sda1.dd for port 2001, fi1001.sda2.dd for port 2002, and fi1001.sda5.dd for port 2005. On the source PC, the output of dd was piped to netcat (nc) which was then instructed to send that data to 172.26.1.103 on either port 2001 for /dev/sda1, 2002 for /dev/sda2 and 2005 for /dev/sda5.

Once the copy was complete, sha1sum was run against the newly acquired images on the forensic workstation and compared to those obtained from the compromised machine. With the exception of sda5, since an original hash could not be obtained, they matched.

```
[root@localhost image]# sha1sum *
5e9d839848ca63f42788fcac5e53d7d2b7f9a7b3  fi1001.sda1.dd
70cc8b5c0282e5114fd3f3688571d9623bdf7214  fi1001.sda2.dd
6b7a273cc143e13021250b25cbeedf995b9c6128  fi1001.sda5.dd
```

It should therefore be noted that any data used as evidence obtained from the Linux Swap partition cannot be guaranteed to be exactly the same as it is on the original evidence as a hash could not be produced.

The PC containing the compromised VMWare session was then powered off and tagged as evidence. From this point onward the original evidence was not needed any further, and all work was carried out on the working copies of the partitions acquired on the forensic workstation.

## Producing a Timeline

The next stage was to generate a timeline so specifics on when the files on disk were actually created can be seen, as well as last modified and accessed. To do this was a two stage process. Firstly, the *fls* tool was used to compile data about the files on the sda1 partition image, and then secondly the *mactime* tool was used to take the output of *fls* and put it into a readable timeline format. Both the *fls* and *mactime* tools are part of the The Sleuthkit. Since the *fls* tool requires a filesystem type to be specified, this had to be identified first using the *file* command, which was passed the image of the sda1 partition as a parameter<sup>16</sup>.

```
[root@localhost fi1001]# file fi1001.sda1.dd
fi1001.sda1.dd: Linux rev 1.0 ext 2 filesystem data (mounted or unclean)
```

The original sda1 partition was identified as a linux ext 2 filesystem. *fls* was then run.

```
[root@localhost fi1001]# fls -f linux-ext2 -m / -r fi1001.sda1.dd >
timeline.sda1.fls
```

-f	Specifies the filesystem. This was previously determined using <i>file</i> command.
-m /	Output in mactime format and put a / before every name
-r	Recurse directories
fi1001.sda1.dd	The image we are interrogating
> timeline.sda1.fls	The output file

The timeline.fls file produced contained details of every file, including deleted one (subject to there still being an entry in the inode table for it of course), on the system. The *-m* switch is specified so that it outputted in a format that can then read into the mactime tool which then formatted the data into a more readable format.

```
[root@localhost fi1001]# mactime -b timeline.sda1.fls >
timeline.sda1.mac
```

-b	Specifies the location of the “body” file to read in, which contains the raw data.
timeline.sda1.fls	The file containing the output from the previous fls command.
> timeline.sda1.mac	The output file

---

<sup>16</sup> Running the command with *--help* provides a list of supported file systems and the parameter to be passed for each one.

Once run, the timeline.sda1.mac file contained the timeline in a more readable format.

Since the timeline contains an entry for every single file on the partition, it has been included within a separate document - (MD5: 5596485e4266fa07f5713533a0978c1e) . Any significant references will be reproduced in this report.

### Reading the timeline

The timeline is formatted as follows.

First column contains the date and time that refers to the change, whether it be modify, access, or create, or any combination of all three.

The second column shows the size of the file on disk.

The third column details as to what the entry relates to.

m	Modified
a	Access
c	Create

Modify means the last time the file was written to.

Access refers to the time at which the file itself was last read in some form. This may just be the header, not necessarily the entire file.

Create relates to last time the directory entry was updated. A create time could be updated as a result of a file being renamed for example, as well as its original creation of course.

The fourth column displays the permissions of the file. This is split in to three sections, owner; group; and the world. Each section contains a placeholder for three settings, (r)ead, (w)rite, and e(x)ecute. As an example, a file which has all permissions for the owner, only read permissions for the group, and only execute permissions for everyone else would look like `rwxr---x`.

The fifth column details the owner, and sixth details the group.

Seventh column provides the directory entry or inode number.

And the final column is the filename itself.

## ***Timeline Analysis***

By analysing the timeline, a picture of the activity that took place on the system can be built up. It should be realised that times indicated within the timeline are the last times the activity occurred. The access time for a file, for example, is the last time the file was accessed. It is not possible to determine any times prior that the file was also accessed.

For this reason, while a timeline can be very useful, a lot of analysis is open to interpretation and does not always give a full picture of all activity.

The earliest timestamp recorded on the timeline dates back to April 1985. It may have been an old version of RedHat that was installed, but it's not that old and this is not an indication that it was actually installed in 1985.

During the base build, files are copied across onto the new machine from the source media and the original MAC times are kept as is. By studying the files and the dates recorded it can be safely assumed that anything up until November 3<sup>rd</sup> 2004 does not have correct MAC times associated with the files.

A good indication as to when a Linux operating system was installed is to look for the creation of a large number of devices. Since devices are machine dependant they are created fresh during the installation.

Looking at the timeline, a large number of files were created in /dev stating on November 3<sup>rd</sup> 2004 at 14:36:08. A good one to look for and which is on every single system is /dev/null. On this image it is shown as being created at 14:36:22.

There was also absolutely no activity between March 10<sup>th</sup> 2000 and November 3<sup>rd</sup> 2004, which is quite a large amount of time for nothing to have happened, so that also suggests that November 3<sup>rd</sup> is the correct date.

The installation appears to have finished at 15:26:53 as that marks the end of the consecutive activity and no more is recorded until 05:08:19 the next day.

The following activity seems to indicate the last time the system was booted. At 05:08:25 a number of files in /etc/rc3.d<sup>17</sup> were last accessed, and 05:08:50 the last access time of /etc/rc.d/rc.local<sup>18</sup> is recorded. All these files are generally accessed as part of the boot sequence, but of course this is not conclusive. A number of other files that are typically only used when the system is loading also have their last access time at around this time, which enhances the evidence that these times are as a result of a system boot.

Since there is no activity between what has been recognised as activity during installation and activity during booting this also suggests that the system has only ever been booted once.

There is a large time gap between 05:13:21 and 20:15:56, so this would mark the end of the system being booted.

---

<sup>17</sup> The /etc/rc directories contain links to the start-up scripts of services which are run when the system is booted.

<sup>18</sup> The /etc/rc.d/init.d file is read during boot-up and commands within it are executed.

### Sat Dec 16 1989

A number of files within subdirectories of the ch0pper home directory are seen. Obviously these times are not accurate and may be a result of MAC time tampering, but more likely due to them being extracted from a tar file and the original attributes being retained.

```
Sat Dec 16 1989 19:21:07
5212 m.. -/-r--r--r-- 1000      101      379369
/home/ch0pper/lrk5/sniffer/libpcap-0.4/SUNOS4/nit_if.o.sparc
Sat Dec 16 1989 19:21:14
4267 m.. -/-r--r--r-- 1000      101      379370
/home/ch0pper/lrk5/sniffer/libpcap-0.4/SUNOS4/nit_if.o.sun3
Wed Mar 21 1990 10:21:26
5291 m.. -/-r--r--r-- 1000      101      379368
/home/ch0pper/lrk5/sniffer/libpcap-0.4/SUNOS4/nit_if.o.sun4c.4.0.3c
```

The reference to lrk5 suggests that the files are part of the Lrk5 rootkit. Further references to files located within the Lrk5 directory under ch0pper are seen within the timeline with old and inaccurate modify times.

### Sat Nov 06 2004

Two deleted files with recorded last access times of 12:08:46. The name of the directory they are located in is t0rnkit2 which is the name of another rootkit, bringing the total number of rootkits possibly installed on this system to at least 2. Again, this is also within the ch0pper home directory tree.

```
Sat Nov 06 2004 12:08:46
197184 .a. -/-r-sr-xr-x 501      501      493481
/home/ch0pper/t0rnkit2/in.rshd (deleted-realloc)
7804 .a. -/-rwxr-xr-x 501      501      493479
/home/ch0pper/t0rnkit2/in.fingerd (deleted-realloc).
```

The *(deleted-realloc)* reference indicates that the file has been deleted.

### Sun Nov 07 2004

At 17:48:00 last access times of .bash\_profile, .bashrc and .bash\_history in /home/test/ are recorded. This is closely followed by access of .bash\_logout and modify/create times of .bash\_history being updated. This suggests that the account test logged in at 17:48:00 and logged out at 17:48:51.

Between these times, the access time for /usr/bin/uptime was updated. The uptime command displays details about how long a system has been running, the number of logged in users, and the current load on the server.

```
Sun Nov 07 2004 17:48:00
230 .a. -/-rw-r--r-- 500      500      84729      /home/test/.bash_profile
124 .a. -/-rw-r--r-- 500      500      84730      /home/test/.bashrc
73 .a. -/-rw----- 500      500      84734      /home/test/.bash_history
Sun Nov 07 2004 17:48:21
2836 .a. -/-r-xr-xr-x 0        0        246052      /usr/bin/uptime
```

```
Sun Nov 07 2004 17:48:51
24 .a. -/-rw-r--r-- 500      500      84728      /home/test/.bash_logout
73 m.c -/-rw----- 500      500      84734      /home/test/.bash_history
```

### Sat Nov 13 2004 – 19:37

Modify and Create time for /home/ch0pper/Desktop changed. Various other files within /home/ch0pper are seen with create and modify times set at this time. As many of the files are typically created by default whenever a new account is created, this indicates that not only was the ch0pper user created at this time, but also it was created using normal methods such as with the useradd command, and not created manually by editing and creating files.

```
Sat Nov 13 2004 19:37:21
4096 m.c d/drwxr-xr-x 501      501      445180
/home/ch0pper/Desktop
24 mac -/-rw-r--r-- 501      501      477950
/home/ch0pper/.bash_logout
124 m.c -/-rw-r--r-- 501      501      477952
/home/ch0pper/.bashrc
230 m.c -/-rw-r--r-- 501      501      477951
/home/ch0pper/.bash_profile
```

Last access time for /usr/sbin/useradd is updated which confirms that useradd was used to create the user and the associated files/directories. Additionally, /etc/group has modify and create times set.

```
Sat Nov 13 2004 19:37:21
53200 .a. -/-rwxr-xr-x 0      0      49314      /usr/sbin/useradd
527 m.c -/-rw-r--r-- 0      0      198599      /etc/group
```

Access time on /usr/bin/passwd is updated. This command is used for the updating of a users password. Since /etc/passwd and /etc/shadow modify times are then subsequently updated this shows that a password was changed.

```
Sat Nov 13 2004 19:37:39
12244 .a. -/-r-s--x--x 0      0      247434      /usr/bin/passwd
Sat Nov 13 2004 19:38:02
919 m.c -/-rw-r--r-- 0      0      199499      /etc/passwd
Sat Nov 13 2004 19:38:03
814 m.c -/-r----- 0      0      199506      /etc/shadow
```

### Sat Nov 13 - 20:21

A deleted file within /home/ch0pper is recorded as last accessed. Listing makes reference to t0rnkit2/keffimap.

```
Sat Nov 13 2004 20:21:28
478312 .a. -/-rwxr-xr-x 501      501      493484
/home/ch0pper/t0rnkit2/keffimap (deleted-realloc)
```

### Sat Nov 13 – 20:43

Modify and Create time set on /home/ch0pper/t0rnkit.tar. From the filename this suggests this is the source archive for the t0rn rootkit, and it was created on this system at this time.

At 20:52:55, several files were created under the /home/ch0pper/t0rnkit2 directory, which could be as a result of the t0rnkit.tar archive being extracted. As the last access time of t0rnkit.tar is at 20:52:56 this would be very probable.

```
Sat Nov 13 2004 20:43:48
1407746 m.c -/-rw-r--r-- 501      501      477955
/home/ch0pper/t0rnkit.tar
Sat Nov 13 2004 20:52:55
18 ..c -/-rw-r--r-- 501      501      379019
/home/ch0pper/t0rnkit2/dev/.laddr
20 ..c -/-rw-r--r-- 501      501      379021
/home/ch0pper/t0rnkit2/dev/.lproc
Sat Nov 13 2004 20:52:56
1407746 .a. -/-rw-r--r-- 501      501      477955
/home/ch0pper/t0rnkit.tar
```

At 20:56:03 the modify and create time on /var/log/xferlog is updated. This log file records details of ftp transfers. At the same time modify and create times on lrk5.tar.src.tar in /home/ch0pper are changed. This file is likely to be the source tar archive for the lrk5 rootkit, and the update of the xferlog points to it being uploaded via ftp.

```
Sat Nov 13 2004 20:56:03
211 m.c -/-rw----- 0          0      232210 /var/log/xferlog
3301054 m.c -/-rw-r--r-- 501      501      477957
/home/ch0pper/lrk5.src.tar.tar
```

At 20:56:11 the directory /dev/sdc0 is created, which indicates that the installation of a rootkit has occurred. Rootkits typically will try and hide themselves within the /dev directory as there is an awful lot in there and quite frankly it's a good place to hide. A number of other files are then seen to be created under this directory, including a file called t0rn sniff which is likely to be associated with a packet sniffer.

```
Sat Nov 13 2004 20:56:11
4096 m.c d/drwxr-xr-x 0          0      117502 /dev/sdc0
3215 ..c -/-rwxr-xr-x 501      501      493477
/dev/sdc0/.nfs01/hidemodule.c
407450 ..c -/-rwx--x--x 501      501      493473
/dev/sdc0/.nfs01/sshbd.tgz
4096 m.c d/drwxr-xr-x 0          0      461566 /dev/sdc0/.nfs01
2749 ..c -/-rwxr-xr-x 501      501      493476
/dev/sdc0/.nfs01/phide.tgz
9361 ..c -/-rwxr-xr-x 501      501      493489
/dev/sdc0/.nfs01/t0rn sniff
```

The create times of a number of core system binaries such as ls, ps, netstat are then changed. The following binaries are now likely to be trojan versions designed to provide a backdoor into the system and hide the presence of the rootkit and anything that may raise suspicion.

/usr/sbin/in.inetd	/usr/bin/top	/bin/netstat
/bin/login	/usr/sbin/in.rshd	/usr/bin/rlogin
/usr/sbin/smbd	/usr/bin/du	/bin/ls
/bin/ps	/usr/sbin/in.fingerd	/usr/sbin/imapd
/usr/sbin/named	/usr/sbin/rpc.mountd	

Since there was a lot of access activity taking place in the /home/ch0pper/t0rnkit2 directory while these new binaries were being written, it appears that the t0rnkit rootkit was successfully installed.

The last access time on /home/ch0pper/t0rnkit2/t0rn, which is the installation script for the t0rn rootkit is also recorded.

```
Sat Nov 13 2004 20:56:12
6889 .a. -/-rwxr-xr-x 501      501      493475
/home/ch0pper/t0rnkit2/t0rn
```

#### **Sat Nov 13 2004 - 20:49**

File under the ch0pper home directory called ch0pper.sh is created, modified and accessed. This file should be investigated further in order to identify the contents.

```
Sat Nov 13 2004 20:49:11
26 ma. -/-rwxr-xr-x 501      501      477959  /home/ch0pper/ch0pper.sh
Sat Nov 13 2004 20:49:18
26 ..c -/-rwxr-xr-x 501      501      477959  /home/ch0pper/ch0pper.sh
```

#### **Sat Nov 13 2004 - 21:37**

/home/ch0pper/lrk5 directory is created and a lot of create and access activity in the /home/ch0pper/lrk5 directory is observed. Likely cause is the extraction of the lrk5.src.tar.tar archive that was previously observed being created at 20:56:03

```
Sat Nov 13 2004 21:37:07
4096 ..c d/drwxr-xr-x 1000     101      428798  /home/ch0pper/lrk5
```

#### **Sat Nov 13 2004 - 21:39**

Last access time for /usr/bin/make is set. *make* is typically used when compiling new packages, and will most probably have been used when compiling the rootkits. Since there is no evidence of any binaries being created after this time, and the last create time of key binaries that are usually overwritten by rootkits



such as ls, ps, etc remain at the time when the t0rnkit was installed it is likely that whatever was being "made" failed.

```
Sat Nov 13 2004 21:39:13
111472 .a. -/-rwxr-xr-x 0          247217  /usr/bin/make
```

#### **Sat Nov 13 2004 - 21:48**

The modify and create times for /home/httpd/html/index.html are set, which is the default Apache test page which is served. It is known that this page was defaced by the attacker, and the create and modify times set suggest that this is when it occurred.

```
Sat Nov 13 2004 21:48:57
2579 m.c -/-rw-r--r-- 0          82110
/home/httpd/html/index.html
```

After this time there are a large number of access times updated on a wide variety of files, but no significant modify or create times are observed. This would therefore mark the end of changes to the system by the attacker.

## Log File Analysis

Since a rootkit has been installed, it should also be noted that some will modify system log files in order to hide any evidence that an attacker has entered the system. This is typically done by examining the current IP address and removing any entries from log files that make reference to that IP address.

In order to view the log files the filesystem was mounted on the forensics workstation as read only to ensure that no modification of the image could take place.

```
[root@localhost fi1001]# mkdir /dev/fi1001
[root@localhost fi1001]# mount -o ro,loop fi1001.sda1.dd /mnt/fi1001
```

-o	Specifies there are options to follow
ro	Mount as read-only.
loop	Use the loop device <sup>19</sup>
fi1001.sda1.dd	Our image file we wish to mount
/mnt/fi1001	The mount point where we can access it on our filesystem

First port of call was the /var/log/messages<sup>20</sup> file.

In it a number of interesting entries were discovered.

```
Nov 13 19:37:21 localhost useradd[16153]: new group: name=ch0pper,
gid=501
Nov 13 19:37:21 localhost useradd[16153]: new user: name=ch0pper,
uid=501, gid=5
01, home=/home/ch0pper, shell=/bin/bash
Nov 13 19:37:21 localhost useradd[16153]: add `ch0pper' to group `root'
Nov 13 19:37:21 localhost useradd[16153]: add `ch0pper' to shadow group
`root'
Nov 13 19:38:03 localhost PAM_pwd[16154]: password for (ch0pper/501)
changed by
  (null)/0)
```

At 19:37:21 on November 13<sup>th</sup> several entries were found detailing the creation of a new user called ch0pper, who's name has already been previously highlighted on the defacement itself, not to mention also having two rootkits in his/her home directory.

The log also shows the ch0pper user being added to the root group. At 19:38:03 the password for the user was changed. This matches with a one second difference the entry in the timeline highlighting /etc/passwd being modified.

At 19:38:32 and 20:21:47 user ch0pper is recorded logging into the system.

---

<sup>19</sup> The loop device allows you to mount an image of a file system stored as a standard file as though it was an external disk.

<sup>20</sup> most applications will tend to write general log file entries to /var/log/messages

```
Nov 13 19:38:32 localhost PAM_pwdb[16156]: (login) session opened for
user ch0pper by (uid=0)
Nov 13 20:21:47 localhost PAM_pwdb[16228]: (login) session opened for
user ch0pper by (uid=0)
```

Taking a look at /var/log/secure showed the following two entries related to ch0pper which also provides us with a source ip address.

```
Nov 13 19:38:32 localhost login: LOGIN ON 0 BY ch0pper FROM
81.146.41.184
Nov 13 20:21:47 localhost login: LOGIN ON 1 BY ch0pper FROM
81.146.45.181
```

From this it can be seen that ch0pper has logged in from two different IP addresses, 81.146.41.184 and 81.146.45.181  
Performing a lookup on these addresses in the RIPE<sup>21</sup> database provides the following information.

```
whois -h whois.ripe.net 81.146.41.184
% This is the RIPE Whois tertiary server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/db/copyright.html
```

```
inetnum:      81.146.24.0 - 81.146.73.255
netname:      REMOTE-INTERNET-COMPLETE
descr:        BT WebPort Complete
descr:        Broadband
country:      GB
admin-c:      MK3517-RIPE
tech-c:       RJD2-RIPE
status:       ASSIGNED PA
remarks:      Please send abuse
notification to abuse@bt.net
mnt-by:       BTNET-MNT
mnt-lower:    BTNET-MNT
mnt-routes:   BTNET-MNT
changed:      preston.dialip@bt.com
20030924
source:       RIPE
```

```
route:        81.128.0.0/11
descr:        BT Public Internet Service
origin:       AS2856
mnt-by:       BTNET-MNT
changed:      support@bt.net 20030615
source:       RIPE
```

```
person:       Mark Kendall
address:      pp 411/A4
address:      Anzani House
address:      Trinity Avenue
address:      Felixstowe
address:      Suffolk
address:      IP11 4XB
address:      U.K.
```

```
whois -h whois.ripe.net 81.146.45.181
% This is the RIPE Whois tertiary server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/db/copyright.html
```

```
inetnum:      81.146.24.0 - 81.146.73.255
netname:      REMOTE-INTERNET-COMPLETE
descr:        BT WebPort Complete
descr:        Broadband
country:      GB
admin-c:      MK3517-RIPE
tech-c:       RJD2-RIPE
status:       ASSIGNED PA
remarks:      Please send abuse
notification to abuse@bt.net
mnt-by:       BTNET-MNT
mnt-lower:    BTNET-MNT
mnt-routes:   BTNET-MNT
changed:      preston.dialip@bt.com
20030924
source:       RIPE
```

```
route:        81.128.0.0/11
descr:        BT Public Internet Service
origin:       AS2856
mnt-by:       BTNET-MNT
changed:      support@bt.net 20030615
source:       RIPE
```

```
person:       Mark Kendall
address:      pp 411/A4
address:      Anzani House
address:      Trinity Avenue
address:      Felixstowe
address:      Suffolk
address:      IP11 4XB
address:      U.K.
```

<sup>21</sup> The RIPE database contains details of who owns what IP addresses within Europe and the Middle East (other areas are also covered, but these are the main regions). Three other databases exist; ARIN, APNIC and LACNIC which hold IP address information for other parts of the world.

```

phone:      +44 1394 601 589
fax-no:     +44 1394 273 463
e-mail:     mark.kendall@bt.com
nic-hdl:    MK3517-RIPE
changed:    preston.dialip@bt.com
20030320
source:     RIPE

person:     Bob Dootson
address:    pp HWC473
address:    Virtual Postbox (HOM-NZ)
address:    PO Box 200
address:    London
address:    N18 1ZF
address:    U.K.
phone:      +44 1977 597630
fax-no:     +44 1943 468450
e-mail:     bob.dootson@bt.com
nic-hdl:    RJD2-RIPE
changed:    preston.dialip@bt.com
20030320
source:     RIPE

```

```

phone:      +44 1394 601 589
fax-no:     +44 1394 273 463
e-mail:     mark.kendall@bt.com
nic-hdl:    MK3517-RIPE
changed:    preston.dialip@bt.com
20030320
source:     RIPE

person:     Bob Dootson
address:    pp HWC473
address:    Virtual Postbox (HOM-NZ)
address:    PO Box 200
address:    London
address:    N18 1ZF
address:    U.K.
phone:      +44 1977 597630
fax-no:     +44 1943 468450
e-mail:     bob.dootson@bt.com
nic-hdl:    RJD2-RIPE
changed:    preston.dialip@bt.com
20030320
source:     RIPE

```

Records for both IP addresses are identical, and both IP addresses do fall within the IP address range specified by the inetnum field on both.

```
inetnum:      81.146.24.0 - 81.146.73.255
```

The description given within each record shows that the IP addresses are owned by BT and suggests that they are provided for use by Broadband customers. It is common for a Broadband customer to be provided with a different IP address whenever their connection is reset, either from the remote end or because of their local router/modem being reset.

This being the case it suggests that sometime between 19:38:32 and 20:21:47 the internet connection used by ch0pper was reset, and both IP addresses recorded are the same individual. Going forward this should be remembered, as connections from the attacker may not be just limited to these two listed IP addresses.

A court order being presented to BT requesting user information on these connections will provide details of the actual person responsible, or at least the person who pays for the broadband account.

A further search of /var/log/secure and /var/log/messages was performed and compared for any IP address within the established range. On November 10<sup>th</sup> at 04:16:33 a telnet connection was observed and at 04:16:52 a failed login attempt was recorded originating from 81.146.25.222. No user name was provided.

The reason for the 19 second gap between the two entries could be a result of either a slow system, or it took 19 seconds for the person connecting to enter a username and password.

Please the note the following key is used when referring log file entries.

- [1] refers to an entry in /var/log/secure
- [2] refers to an entry in /var/log/messages

```
[1] Nov 10 04:16:33 localhost in.telnetd[12440]: connect from
81.146.25.222
[2] Nov 10 04:16:52 localhost login: FAILED LOGIN 1 FROM 81.146.25.222
FOR , User not known to the underlying authentication module
```

Starting at 07:12:14 on November 13<sup>th</sup> and going through until 07:12:54 around 60 failed login attempts for root from 81.146.41.184 were recorded.

```
[1] Nov 13 07:12:06 localhost in.telnetd[15384]: connect from
81.146.41.184
[2] Nov 13 07:12:14 localhost login[15396]: FAILED LOGIN 1 FROM
81.146.41.184 FOR root, Authentication failure
.
.
.
[1]
Nov 13 07:12:06 localhost in.telnetd[15384]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15385]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15386]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15387]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15394]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15393]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15392]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15391]: connect from 81.146.41.184
Nov 13 07:12:07 localhost in.telnetd[15390]: connect from 81.146.41.184
```

Due to the frequency of the telnet connections and the subsequent reported failures, amounting to several each second, it is presumed that this is a result of an automated brute force attack as a human is physically not able to type this fast.

The reason the brute force attack only lasted a short amount of time is because the telnet service reported at 07:12:48 in /var/log/messages it was going to cease accepting connections due to being overloaded.

```
[2] Nov 13 07:12:48 localhost inetd[530]: telnet/tcp server failing
(looping or being flooded), service terminated for 10 min.
```

There are no successful login attempts recorded, so it is clear the attempt to brute force the root password was not successful.

Between 18:41:24 and 19:28:43 several ftp connects were observed.

On the first connection, ftp login using user root failed.

```
[1] Nov 13 18:41:24 localhost in.ftpd[15946]: connect from 81.146.41.184
[2] Nov 13 18:42:49 localhost ftpd[15946]: PAM-listfile: Refused user
root for service ftp
```

An attempt to log in using a user account that did not exist on the system was then recorded. Unfortunately in instances where the user account is not valid, the passed credentials are not logged.

```
[1] Nov 13 18:50:41 localhost in.ftpd[15950]: connect from 81.146.41.184
[2] Nov 13 18:51:02 localhost PAM_pwdb[15950]: check pass; user unknown
```

At 18:51:35 there was a successful anonymous ftp login. During an anonymous login the user is asked to give their email address as the password for auditing purposes. Of course there is nothing to prevent a fake address being given, and even during legitimate anonymous logins most of the Internet population never do give their real email address. In this instance, the email address fred@fred.com was passed.

```
[1] Nov 13 18:51:35 localhost in.ftpd[15951]: connect from 81.146.41.184
[2] Nov 13 18:51:49 localhost ftpd[15951]: ANONYMOUS FTP LOGIN FROM
81.146.41.184 [81.146.41.184], fred@fred.com
```

A number of further anonymous ftp logins were observed, each time the email address fred@fred.com<sup>22</sup> was given as the password. Examining the /var/log/xferlog log file gave no more indications as to what happened during the logged in anonymous ftp sessions, although the entire contents of the log contained just two lines.

```
Sat Nov 13 20:43:48 2004 306 81.146.45.181 1407746
/home/ch0pper/t0rnkit.tar b _ i r ch0pper ftp 0 * c
Sat Nov 13 20:56:03 2004 596 81.146.45.181 3301054
/home/ch0pper/lrk5.src.tar.tar b _ i r ch0pper ftp 0 * c
```

This shows how the root kits were transferred to the compromised system then. t0rnkit was transferred at 20:43:48 on Saturday November 13<sup>th</sup> 2004. lrk5 was transferred by the ch0pper user at 20:56:03 also on Saturday November 13<sup>th</sup> 2004.

Again note that the IP address recorded was 81.146.45.181, whereas the anonymous ftp logins were from 81.146.41.184.

A look at the /var/log/httpd/access.log and /var/log/httpd/error.log files showed if there were any accesses by the attacker to the http web server, and there was. From the timestamps on the log entries it was clear that the majority of the requests being made were via an automated tool as there were several a second. A browse through the entries also showed that the requests were for common vulnerable scripts or files that could either be used to gain entry to the system or provide more information. None of these scripts or files existed on the system, so would have not presented a potential route.

```
[Sat Nov 13 07:34:01 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/.DS_Store
[Sat Nov 13 07:34:02 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/.FBCIndex
```

---

<sup>22</sup> fred@fred.com is a common address used as an example, like test@test.com and someone@somewhere.com. It is unlikely to be the actual email address of the attacker.

```
[Sat Nov 13 07:34:04 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/admin.cgi
[Sat Nov 13 07:34:06 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/docs/
[Sat Nov 13 07:34:07 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/index.html.ca
[Sat Nov 13 07:34:08 2004] [error] [client 81.146.41.184] File does not
exist: /home/httpd/html/index.html.cz.iso8859-2
```

The first entry from one of the already identified IP addresses in `/var/log/httpd/access.log` on the 13<sup>th</sup> November is at 07:25:14 and is not a valid http request that would be sent from a normal web browser as it is just "get" and does not contain any details as to the protocol used or what page is being requested. This request was typed by hand, and was an attempt to try to determine what version of web server is running

```
81.146.41.184 - - [13/Nov/2004:07:25:14] "get" 501 -
```

At this stage, however, there is no evidence of how the attacker gained entry to the system, only that he/she did.

Each user has a history file located within their home directory which contains a complete audit of all commands typed. Since the default shell on Linux is bash, this file is typically called `.bash_history`.

A `.bash_history` file was found in `/home/ch0pper`, and contains a list of all commands the attacker typed while on the system. This gives a great insight into what has occurred, and when compared with the file timeline a possible date and time some of the commands were typed can be estimated.

Below is a listing of the `.bash_history` file for the `ch0pper` user along with comments on what may be occurring.

Command	Time	Comments
<code>exit</code>		Since <code>exit</code> appears as the first command typed, there are two possibilities. The first is that immediately after the attacker gained access to the system using the <code>ch0pper</code> account they exited, or that any previous commands within the history file have been erased.
<code>cd /var/www/html</code>		Change of directory to <code>/var/www/html</code> . Newer versions of apache invite the administrators to store all web content under this directory. Since this is an old version of Apache, the default directory is actually <code>/home/httpd/html</code> . Changing to this directory was therefore unsuccessful as the directory did not exist on this system



cd /var/www		As /var/www/html did not exist, the attacker attempted to change to /var/www instead. Again, this would have failed.
cd /		Change to the root directory.
ls		Perform a directory listing.
locate -u		Locate (or slocate) is used to find files within the system. slocate uses a database in order to provide quick searching, and this database is updated by providing the -u switch.
locate index.htm*		Attacker attempts to locate the <i>index.htm</i> file. Since subsequent commands do not suggest the attacker is heading towards the location of this file, it can be assumed that the use of locate failed to provide the information needed.
cd /etc/		Change to the /etc directory
ls htt*		List all files beginning with <i>htt</i>
more php3.ini		List the contents of the <i>php3.ini</i> file. Since the previous ls actually listed the contents of the httpd directory and not the current one, this operation failed.
pwd		Display what the current directory is
ls   more		Perform a directory listing
more httpd*	13 Nov 19:53	List the contents of all files starting with <i>httpd</i>
cd httpd		Change into the <i>httpd</i> directory
ls		Perform a directory listing
cd conf		Change into the <i>conf</i> directory
ls		Perform a directory listing
more httpd.conf		List the contents of the <i>httpd.conf</i> file. This is the main Apache configuration file and within it would be listed details of where the web root is, and therefore details of where the main <i>index.html</i> file is.
ls		Perform a directory listing
pwd		Display what the current directory is
cd ..		Change back one directory to <i>/etc/httpd</i>
ls		Perform a directory listing
more php3.ini	13 Nov 19:56	List the contents of the <i>php3.ini</i> file.



ls		Perform a directory listing
pwd		Display what the current directory is
find / -name *.htm* -print		Find all files with a <i>.htm</i> extension. Since the attacker is now doing search for html files it seems that they did not find the information they were looking for in the <i>httpd.conf</i> file.
cd /home/httpd/html		Change to <i>/home/httpd/html</i> directory. This is where the default <i>index.html</i> file is located, so the previous find command was successful in providing the information the attacker required.
ls		Perform a directory listing
more index.html		List the contents of the <i>index.html</i> file, which is the default Apache test page that is being presented to visitors.
cp index.html index.sav		Make a copy of the <i>index.html</i> file and save it as <i>index.sav</i> . This is very strange. The only instance where somebody makes a backup copy of a file is if they plan to replace any changes they make with the original again. If you are planning to deface a website, why would you need to make a backup of the original source file? <i>index.sav</i> does not appear on the timeline, and has not been recovered as a deleted file, so this suggests that the copy operation may have failed.
ls -la		Perform a directory listing, listing all files including hidden
chmod 777 index.html		Change the permissions on <i>index.html</i> to be full to owner, group and world.
ls -la		Perform a directory listing
pwd		Display what the current directory is
ls /etc/wu*		Perform a directory listing of anything starting with <i>wu</i> in <i>/etc/</i>

It is known that a rootkit has been installed, and the *index.html* file was modified, but there is no evidence of this within the *.bash\_history* file of *ch0pper*.

Looking at the contents of the *.bash\_history* file for the root user gave the following. Since the root user is legitimate and would have been used by the system administrator it is unclear however which commands were actually typed

by the attacker. As this is a honeypot though, it is known what commands have been typed by legitimate users.

Command	Time	Comments
<code>ifconfig</code>		Display details about the network configuration of the system
<code>shutdown -h now</code>		Shutdown the system. Since the system was still running this was probably not typed by the attacker.
<code>date</code>		Display the date and time
<code>date -s 20:15</code>		Set the time
<code>exit</code>		exit
<code>cd /home/ch0pper</code>		Change to the ch0pper directory. Since this directory did not exist before the attack, this is likely to be the first command typed by the attacker
<code>cd t0rnkit2/</code>		Change into the t0rnkit2 directory. In order to create the t0rnkit2 directory in the first place the tar archive would have to be extracted. There is no evidence of the command used to perform that operation.
<code>./t0rn</code>	13 <sup>th</sup> Nov 20:56	Run the t0rn installation script.
<code>fingerd</code>		Run the finger daemon.
<code>id</code>		Display details of the current user.
<code>/sbin/in.fingerd</code>		This file does not exist on this system
<code>cd ..</code>		Change back a directory to /home/ch0pper
<code>ls</code>		Perform a directory listing
<code>tar -zxvf lrk5.src.tar.tar</code>	13 <sup>th</sup> Nov 21:37	Extract the lrk5.src.tar.tar file
<code>cd lrk5</code>		Change into the lrk5 directory that was created as a result of the extraction
<code>ls</code>		Perform a directory listing
<code>./configure</code>		Run the configure script. This prepares for the installation.
<code>./configure -n</code>	13 <sup>th</sup> Nov 21:37	Run the configure script with the -n flag.
<code>more README</code>	13 <sup>th</sup> Nov 21:41	List the contents of the README file. Reading a README file is an indication that the attacker was not confident with the task he/she was performing, and required help.
<code>make all install</code>		Run the make command to install the rootkit.

ls		Perform a directory listing
make		Run make to prepare for installation
make install	13 <sup>th</sup> Nov 21:39	Run make install
cd ..		Change back a directory to /home/ch0pper
ls		Perform a directory listing
cd t0rnkit2		Change into the t0rnkit2 directory
ls		Perform a directory listing
who		Run who to display who is on the system
more README	13 <sup>th</sup> Nov 21:41	View the README file
ls		Perform a directory listing
cd t0rn		Attempt to change into the t0rn directory. Since t0rn in /home/ch0pper/t0rnkit2 is actually a file and not a directory, the command would have failed.
cd ..		Change back a directory to /home/ch0pper/
cd ..		Change back a directory to /home/
ls		Perform a directory listing
cd httpd		Change into the httpd directory. This directory does not exist, and is likely a typo.
cd httpd		Change into the /home/httpd directory
ls		Perform a directory listing
cd html		Change into the /home/httpd/html directory.
ls		Perform a directory listing
vi index.html		Edit the index.html file. This will most probably be where the file was modified in order to deface the website.
vi index.html	13 <sup>th</sup> Nov 21:48	Edit the index.html file again.
ls /home/ch0pper		Perform a directory listing of /home/ch0pper
nc -vv -l -p 9999 -e /home/ch0pper/ch0pper.sh		Run a netcat listener on port 9999 and after a connection run /home/ch0pper/ch0pper.sh Upon running this on another system using the version of nc on the RedHat 6.2 system it was found that nc immediately returned with no output - it did not execute as anticipated
nc -vv -l -p 9999 &	13 <sup>th</sup> Nov 21:50	Run a netcat listener on port 9999 and run in background

ps -aux   more		Display a process listing. The attacker was probably checking to ensure that netcat was still running
exit		Exit
locate in.amqd		Locate the in.amqd file. in.amqd is part of the t0rn rootkit. A search on Google for this file will bring back results referencing the t0rn rootkit.
locate -u		Update the locate database. Since the previous command is run again after the update, it looks like locate failed to locate in.amqd. The timeline did indicate that there was a lot of access activity towards just before the system was brought down, which may have been generated by the use of this command.
locate in.amqd	13 <sup>th</sup> Nov 22:14	Locate the in.amqd file again
exit		Exit.

## File System Analysis

It is known that from the evidence collected so far that the attacker created a user account. Examining the `/etc/passwd` file this confirms that the account still exists.

```
ch0pper:x:501:501::/home/ch0pper:/bin/bash
```

The account has a user and group identifier of 501. Examining the timeline again this information can be used to list all files where either the user or group of ch0pper is classed as the owner.

It is also known that a home directory was also created, as this is where the rootkits were stored.

```
[root@localhost ch0pper]# pwd
/mnt/fil001/home/ch0pper
[root@localhost ch0pper]# ls -l
total 4624
-rwxr-xr-x    1 snort    snort              26 Nov 13 20:49 ch0pper.sh
drwxr-xr-x    5 snort    snort          4096 Nov 13 19:37 Desktop
drwxr-xr-x   21 1000     101          4096 Aug 12 1999 lrk5
-rw-r--r--    1 snort    snort    3301054 Nov 13 20:56 lrk5.src.tar.tar
drwx-----    3 snort    snort          4096 Nov 13 20:56 t0rnkit2
-rw-r--r--    1 snort    snort    1407746 Nov 13 20:43 t0rnkit.tar
```

In the directory listing displayed above of `/home/ch0pper` you can see that the owner and group are recorded as being "snort". This is incorrect. It must be remembered that this is a mounted filesystem and it is being viewed as an extension of the current filesystem of the forensics analysis machine.

On the forensics analysis workstation snort has a user identifier of 501, which is the same as the user ch0pper on the compromised system.

The lrk5 directory, however, has an owner of 1000 and a group of 101. The reason the number identifiers are showing for this directory is because a uid of 1000 and gid of 101 do not exist on the forensic workstation. Examining `/etc/passwd` and `/etc/group` on the compromised system also shows that the uid and gid do not exist on that system either. This suggests that the directory and files contained within were created by a user that either no longer exists, or that when the files were extracted from the tar file the user attributes remained the same as they were from the original machine the tar file was created on.

Studying the date, Aug 12 1999, it is clear that the file attributes have been carried across from another system.

In the ch0pper home directory it can be seen that the original tar files of the two downloaded rootkits are still there.

```
-rw-r--r--    1 snort    snort    3301054 Nov 13 20:56 lrk5.src.tar.tar
-rw-r--r--    1 snort    snort    1407746 Nov 13 20:43 t0rnkit.tar
```

By referring back to the timeline that was produced, we can gather more information.

```

Sat Nov 13 2004 20:56:03
3301054 m.c -/-rw-r--r-- 501      501      477957
/home/ch0pper/lrk5.src.tar.tar

Sat Nov 13 2004 21:37:06
3301054 .a. -/-rw-r--r-- 501      501      477957
/home/ch0pper/lrk5.src.tar.tar

Sat Nov 13 2004 20:43:48
1407746 m.c -/-rw-r--r-- 501      501      477955
/home/ch0pper/t0rnkit.tar

Sat Nov 13 2004 20:52:56
1407746 .a. -/-rw-r--r-- 501      501      477955
/home/ch0pper/t0rnkit.tar

```

The create and modify times of both files correspond with that reported by the normal directory listing, but the last time the files were access can now be seen. These times do not give any details as to when the installation script for each rootkit was run, just a suggestion as to when the tar file was possibly extracted.

Looking inside the lrk5 directory, again the timestamps show dates as far back as November 24<sup>th</sup> 1998. Locating some of these files on the timeline gives the following information.

```

Thu Aug 12 1999 07:00:26
1664 m.. -/-rw-r--r-- 1000     101      428801
/home/ch0pper/lrk5/bindshell.c

Sat Nov 13 2004 21:37:02
1664 .ac -/-rw-r--r-- 1000     101      428801
/home/ch0pper/lrk5/bindshell.c

Sat Nov 13 2004 21:37:06
247 ..c -/-rwxr-xr-x 1000     101      428850
/home/ch0pper/lrk5/configure

Sat Nov 13 2004 21:37:40
247 .a. -/-rwxr-xr-x 1000     101      428850
/home/ch0pper/lrk5/configure

Tue Nov 24 1998 12:27:05
3591 m.. -/-rw-r--r-- 1000     101      428802
/home/ch0pper/lrk5/fix.c

Sat Nov 13 2004 21:37:02
3591 .ac -/-rw-r--r-- 1000     101      428802
/home/ch0pper/lrk5/fix.c

```

From this more realistic times can be seen. Rather than way back in the 90s, the files were created at around 21:37 on November 13<sup>th</sup> 2004. This coincides with the last access time of lrk5.src.tar.tar

To confirm that the two files that were found are tarballs of the rootkits they claim to be, an attempt to locate the same files on the internet using a standard search engine was made.

A copy of the LRK5 rootkit was found within the AntiServer "Rootkit Download Area" at <http://www.antiserver.it/backdoor-rootkit>. While the filename published here, lrk5.src.tar.gz is slightly different from lrk5.src.tar.tar, once downloaded and md5 checksums compared, they were found to be the same and confirms that the file on the compromised system is the advertised LRK5 rootkit.

```
[root@localhost fil101]# md5sum lrk5.src.tar.gz
e18b708650f7dc4cca447df33d09740f  lrk5.src.tar.gz

[root@localhost ch0pper]# md5sum lrk5.src.tar.tar
e18b708650f7dc4cca447df33d09740f  lrk5.src.tar.tar
```

Unfortunately it was not possible to find a file on the Internet that matched the md5 checksum of t0rnkit.tar file found on the system. Examining the contents of the tar archive though, in particular the README file, it advertises itself as being t0rnkit version 2.0.

It was highlighted within the timeline that files were created under /dev/sdc0. Upon examining this directory, only one directory was found under this called .nfs01. The full-stop at the beginning of the name marks this as a hidden directory and would ensure that it's existence would not be shown by a normal ls directory listing, unless the -a flag was provided.

Under the .nfs01 directory a number of files were discovered, which on first glance would appear to belong to the rootkit. t0rnsniff as its name suggests is possibly a sniffer component.

```
[root@localhost .nfs01]# ls -l
total 484
-rwxr-xr-x  1 snort  snort      3215 Sep 14  1999 hidemodule.c
-rwxr-xr-x  1 snort  snort     46726 Sep 10  1999 in.identd
-rwxr-xr-x  1 snort  snort      2749 Sep 14  1999 phide.tgz
-rwxr-xr-x  1 snort  snort      1345 Sep  9  1999 sauber
-rwx--x--x  1 snort  snort    407450 Sep 14  1999 sshbd.tgz
-rwxr-xr-x  1 snort  snort      6232 Sep  9  1999 t0rnparse
-rwxr-xr-x  1 snort  snort      9361 Sep  9  1999 t0rnsniff
```

The file ch0pper.sh within the /home/ch0pper was highlighted when analysing the timeline. Upon examination the file was simple shell script that when run will give an interactive bash shell.

```
[root@localhost ch0pper]# cat ch0pper.sh
#!/bin/bash
/bin/bash -i
```

From the evidence witnessed so far, it is unlikely that any form of time attributes have been tampered with. If this is the case, and the timeline is to be believed, then no start-up scripts appear to have been modified. To be sure though, the following files were manually viewed and analysed. Nothing was found to be out of the ordinary.

/etc/rc.d/rc.local	Run during startup
/etc/rc.d/init.d/*	All files within this directory are startup scripts for installed services.
/etc/rc.d/rc3.d/* /etc/rc.d/rc5.d/*	These directories contain links to other script files. They define what should be run during startup.
/root/.bashrc /root/.bash_profile /root/.bash_logout	Scripts that are run during login and logout for root.

Furthermore, the filesystem was examined for suid and sgid files.

```
[root@localhost fil001]# find /mnt/fil001 -perm -004000 -o -perm -002000
-type f -ls > /home/forensic/fil001/suid.txt
[root@localhost fil001]# cat /home/forensic/fil001/suid.txt
459025      4 -rwxr-sr-x    1 root    root          3860 Mar  9  2000
./sbin/netreport
 68283     32 -rwxr-sr-x    1 root    games         29704 Feb  8  2000
./usr/X11R6/bin/xbill
245887      8 -r-xr-sr-x    1 root    tty           6128 Mar  7  2000
./usr/bin/wall
246395     32 -rwxr-sr-x    1 root    uucp          30352 Feb 28  2000
./usr/bin/gkermi
246440     40 -r-xr-s--x    1 root    games         40112 Feb 11  2000
./usr/bin/gataxx
246441     24 -r-xr-s--x    1 root    games         20692 Feb 11  2000
./usr/bin/glines
246442     72 -r-xr-s--x    1 root    games         67468 Feb 11  2000
./usr/bin/gnibbles
246443     80 -r-xr-s--x    1 root    games         76508 Feb 11  2000
./usr/bin/gnrobots2
246444     56 -r-xr-s--x    1 root    games         52464 Feb 11  2000
./usr/bin/gnome-stones
246445     76 -r-xr-s--x    1 root    games         71296 Feb 11  2000
./usr/bin/gnomine
246446     28 -r-xr-s--x    1 root    games         25908 Feb 11  2000
./usr/bin/gnotravex
246447    236 -r-xr-s--x    1 root    games        234072 Feb 11  2000
./usr/bin/gtali
246448     24 -r-xr-s--x    1 root    games         24028 Feb 11  2000
./usr/bin/gturing
246449     48 -r-xr-s--x    1 root    games         48316 Feb 11  2000
./usr/bin/iagno
246450     48 -r-xr-s--x    1 root    games         45476 Feb 11  2000
./usr/bin/mahjongg
246451     24 -r-xr-s--x    1 root    games         21140 Feb 11  2000
./usr/bin/same-gnome
```



246574	76	-r-xr-sr-x	1	news	news	73144	Mar	3	2000
./usr/bin/inews									
247219	36	-rwxr-sr-x	1	root	man	36192	Mar	1	2000
./usr/bin/man									
247292	172	-rwxr-sr-x	1	root	uucp	168080	Mar	7	2000
./usr/bin/minicom									
247519	12	-rwxr-sr-x	1	root	mail	11620	Feb	8	2000
./usr/bin/lockfile									
247606	24	-rwxr-sr-x	1	root	slocate	24272	Feb	4	2000
./usr/bin/slocate									
247609	48	-rwxr-s---	1	news	news	47536	Feb	8	2000
./usr/bin/slrnpull									
247875	12	-rwxr-sr-x	1	root	tty	8328	Mar	7	2000
./usr/bin/write									
262666	16	-rwxr-sr-x	1	root	mail	15280	Feb	22	2000
./usr/lib/emacs/20.5/i386-redhat-linux-gnu/movemail									
49334	8	-rwxr-sr-x	1	root	utmp	6096	Feb	25	2000
./usr/sbin/utempter									
49624	12	-rwxr-sr-x	1	root	utmp	8792	Feb	22	2000
./usr/sbin/gnome-pty-helper									
50766	28	-rwxr-sr-x	1	root	lp	25064	Feb	15	2000
./usr/sbin/lpc									

Any files with the *suid* bit set will run under the context of the defined owner when executed. Similarly, with the *sgid* bit set the file to run under the context of the group. These files can be identified by the use of *s* where the *x* (eXecute) placeholder is within the permissions field.

Some programs may need this in order to fulfil their purposes. For example, it may need to have complete access to the entire file system or be able to kill off other users processes.

There is, however, a danger with running programs with elevated privileges, as if there is any weakness that can be exploited then an attack could use it elevate their own privileges.

Taking the *ch0pper.sh* script we have just found as an example, which is currently owned by *ch0pper*. When run by a user, this will just provide a normal bash shell with exactly the same privileges as the user currently has. If, however, the attacker had managed to change the owner of *ch0pper.sh* to root and set the *suid* bit, then when run this would have provided the user with a bash shell with same privileges as root.

From the list of files produced that had either the *suid* or *sgid* bit set, none were found to run under the context of root when executed.

## Recovering deleted files

From analysis of the timeline a number of files marked as deleted were highlighted. Deleted files quite often contain valuable information that an unskilled hacker assumes has gone forever, yet they often can aid in providing a clearer picture as to what has happened on a system.

A list of deleted inodes can be obtained by using the *ils* tool.

```
[root@localhost recovered]# ils -r -f linux-ext2
../.. /image/fi1001.sda1.dd
class|host|device|start_time
ils|localhost|../.. /image/fi1001.sda1.dd|1101009733
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_dtime|st_mod
e|st_nlink|st_size|st_block0|st_block1
1|a|0|0|1099463058|1099463058|1099463058|0|0|0|0|0|0
84811|f|0|0|1100353737|1100353673|1100353737|1100353737|100600|0|12288|1
72395|172396
84812|f|0|0|1100353642|1100353673|1100353737|1100353737|100644|0|2576|17
2403|0
199486|f|0|50|1100345883|1100345883|1100345883|1100345883|100600|0|6|408
299|0
199509|f|0|50|1100345841|1100345841|1100345841|1100345841|100600|0|6|408
291|0
199511|f|0|50|1100345841|1100345841|1100345841|1100345841|100600|0|6|408
292|0
199513|f|0|50|1100345841|1100345841|1100345841|1100345841|100600|0|6|408
294|0
199514|f|0|0|1100345841|1100345882|1100345882|1100345882|100644|0|919|40
8296|0
199515|f|0|0|1100345841|1100345883|1100345883|1100345883|100400|0|769|40
8298|0
215852|f|0|21|1100289728|1100355268|1100355277|1100355277|100640|0|85037
0|439298|439299
215855|a|0|0|1099771323|1099771323|1099771323|0|100644|0|0|0|0
245895|f|0|0|952479772|1100350476|1100350571|1100350571|100755|0|43024|4
92694|492695
245905|f|0|0|952479772|952479772|1100350571|1100350571|100755|0|24752|49
2753|492754
246047|f|0|0|952452206|1100350291|1100350571|1100350571|100555|0|60080|4
93159|493160
246051|f|0|0|952452206|952452206|1100350571|1100350571|100555|0|34896|49
3183|493184
247381|f|0|0|952425102|952425102|1100350571|1100350571|100755|0|66736|51
2316|512317
247553|f|0|0|952425189|952425189|1100350571|1100350571|104755|0|10256|51
5210|515211
247843|a|0|0|952424984|1100348494|1100350571|0|100755|0|20452|518593|518
594
362839|f|0|0|1099466794|1099466794|1099466794|1099466794|100644|0|218|77
2645|0
412498|f|0|0|1100350561|1100350561|1100350636|1100350636|100600|0|0|0|0
412499|f|0|0|1100350636|1100350561|1100350636|1100350636|100600|0|482|82
9305|0
```

```
478003|f|501|501|1100355971|1100355971|1100355971|1100355971|100600|0|0|
0|0
```

-f ext2	Specifies the filesystem. This was previously determined using <i>file</i> command.
-r	Display removed inodes
../../image/fi1001.sda1.dd	Location of the image.

The first column provides the inode numbers of the deleted files. Using the inode number, an associated filename can be obtained using the *ffind* tool.

```
84811|f|0|0|1100353737|1100353673|1100353737|1100353737|100600|0|12288|1
72395|172396
```

```
[root@localhost recovered]# ffind -f linux-ext2
../../image/fi1001.sda1.dd 84811
* /home/httpd/html/.index.html.swp
```

-f ext2	Specifies the filesystem. This was previously determined using <i>file</i> command.
../../image/fi1001.sda1.dd	Location of the image.
84811	The inode number

Taking inode 84811 as an example, using *ffind* it is shown that the filename associated with this inode was */home/httpd/html/.index.html.swp*

Furthermore, deleted files may be recovered by using the *icat* tool and passing it the inode number of the file that needs to be recovered.

```
[root@localhost recovered]# icat -f linux-ext2
../../image/fi1001.sda1.dd 84811 > index.html.swp
```

-f ext2	Specifies the filesystem. This was previously determined using <i>file</i> command.
../../image/fi1001.sda1.dd	Location of the image.
84811	The inode number
> index.html.swp	The name of the recovered file.

Since the task of obtaining a filename and recovering the file has to be done on an inode by inode basis, the following perl script was quickly written in order to automate the task.

The script uses the *lls* tool to list all deleted inodes. The output is then parsed to leave only the inode number, to which a filename if any is obtained using the *ffind* tool. These details are written out to a log file called *recovered.list* for later review. The deleted file itself is then recovered and written out to disk. If a filename was available that the resulting file will be called the same as the original, including the path, but with */*'s replaced by *~*'s. If no filename was available, then the filename of the recovered file will be the inode number.

```

#!/usr/bin/perl
# File recovery script
# Written by Mark Read
# Version 0.1 November 2004

open(FILELIST,">recovered.list");
open(ILS,"ils -r -f linux-ext2 ../../image/fil001.sda1.dd|");
while(<ILS>){
    $filename="[none]";
    ($inode,$dump)=split(/\|/, $_);
    if($inode > 0){
        open(FFIND,"ffind -f linux-ext2
../../image/fil001.sda1.dd $inode|");
        while(<FFIND>){
            $filename=$_;
        }
        chomp($filename);
        if($filename eq "inode not currently used"){
            $saveas=$inode;
        }else{
            $saveas=substr($filename,2,length($filename)-2);
            $saveas=~tr /\//~/;
        }

        print FILELIST "$inode - $filename\n";

        system("icat -f linux-ext2 ../../image/fil001.sda1.dd
$inode > $saveas");
    }
}
close(FILELIST);

exit;

```

After running the script, a total of 22 deleted files were recovered, although only 10 had associated filenames.

The contents of the generated recovered.list file was as follows:

```

1 - inode not currently used
84811 - * /home/httpd/html/.index.html.swp
84812 - * /home/httpd/html/index.html~
199486 - * /etc/shadow.lock
199509 - inode not currently used
199511 - * /etc/shadow.lock
199513 - inode not currently used
199514 - inode not currently used
199515 - inode not currently used
215852 - * /var/lock/makewhatis.lock
215855 - * /var/lock/httpd.lock.621
245895 - inode not currently used
245905 - inode not currently used
246047 - inode not currently used
246051 - inode not currently used
247381 - inode not currently used

```

```

247553 - inode not currently used
247843 - inode not currently used
362839 - * /var/tmp/rpm-tmp.88454
412498 - * /var/spool/mqueue/qfUAA16390
412499 - * /var/spool/mqueue/xfUAA16390
478003 - * /home/ch0pper/.Xauthority-c

```

Performing a directory listing provided a list of following recovered files.

```

-rw-r--r--      1 root      root              0 Nov 21 13:09 1
-rw-r--r--      1 root      root             6 Nov 21 13:09 199509
-rw-r--r--      1 root      root             6 Nov 21 13:09 199513
-rw-r--r--      1 root      root            919 Nov 21 13:09 199514
-rw-r--r--      1 root      root            769 Nov 21 13:09 199515
-rw-r--r--      1 root      root           43024 Nov 21 13:09 245895
-rw-r--r--      1 root      root           24752 Nov 21 13:09 245905
-rw-r--r--      1 root      root           60080 Nov 21 13:09 246047
-rw-r--r--      1 root      root           34896 Nov 21 13:09 246051
-rw-r--r--      1 root      root           66736 Nov 21 13:09 247381
-rw-r--r--      1 root      root           10256 Nov 21 13:09 247553
-rw-r--r--      1 root      root           20452 Nov 21 13:09 247843
-rw-r--r--      1 root      root              6 Nov 21 13:09 ~etc~shadow.lock
-rw-r--r--      1 root      root              0 Nov 21 13:09
~home~ch0pper~.Xauthority-c
-rw-r--r--      1 root      root           2576 Nov 21 13:09
~home~httpd~html~index.html~
-rw-r--r--      1 root      root          12288 Nov 21 13:09
~home~httpd~html~.index.html.swp
-rw-r--r--      1 root      root            767 Nov 21 13:09 recovered.list
-rwxrwxrwx      1 root      root            692 Nov 21 13:09 undelete.pl
-rw-r--r--      1 root      root              0 Nov 21 13:09
~var~lock~httpd.lock.621
-rw-r--r--      1 root      root          850370 Nov 21 13:09
~var~lock~makewhatis.lock
-rw-r--r--      1 root      root              0 Nov 21 13:09
~var~spool~mqueue~qfUAA16390
-rw-r--r--      1 root      root            482 Nov 21 13:09
~var~spool~mqueue~xfUAA16390
-rw-r--r--      1 root      root            218 Nov 21 13:09 ~var~tmp~rpm-
tmp.88454

```

Using the file command it can now be seen what type of files have been recovered

```

[root@localhost recovered]# file *
1:                                empty
199509:                           data
199513:                           data
199514:                          ASCII text
199515:                          ASCII text
245895:                          ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped

```

```

245905: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
246047: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
246051: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
247381: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
247553: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
247843: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses
shared libs), stripped
~etc~shadow.lock: data
~home~ch0pper~.Xauthority-c: empty
~home~httpd~html~index.html~: data
~home~httpd~html~.index.html.swp: data
recovered.list: ASCII text
undelete.pl: perl script text executable
~var~lock~httpd.lock.621: empty
~var~lock~makewhatis.lock: data
~var~spool~mqueue~qfUAA16390: empty
~var~spool~mqueue~xfUAA16390: ASCII text
~var~tmp~rpm~tmp.88454: ASCII text

```

Some of the files recovered had a size of zero bytes, so could not be analysed further. Those files where file reported them as being ASCII were viewed using a standard text editor.

199514 and 199515 contained copies of the current /etc/passwd and /etc/shadow files. Performing a MD5 hash of 199514 and /etc/passwd on the compromised server showed them to be the same.

```

[root@localhost recovered]# md5sum 199514
77a8ce86989c34594927f0cc31ce6914 199514
[root@localhost recovered]# md5sum /mnt/fil001/etc/passwd
77a8ce86989c34594927f0cc31ce6914 /mnt/fil001/etc/passwd

```

The hashes of 199515 and /etc/shadow did not match however, and on closer inspection the difference observed was that in the recovered file no password existed for the ch0pper user.

```

199515:
ch0pper:!!:12735:0:99999:7:::

/mnt/fil001/etc/shadow:
ch0pper:$1$sOx.mMEL$ravPj3S5/0x40UoNHJzWM0:12735:0:99999:7:-1:-
1:134540332

```

Using the *istat* tool, further information can be obtained. Here we see the file was deleted at 19:38:03 on Saturday November 13<sup>th</sup> 2004, and therefore record that the password was changed at this time.

```
[root@localhost recovered]# istat -f linux-ext2
../..image/fil001.sda1.dd 199515
inode: 199515
Not Allocated
Group: 12
uid / gid: 0 / 0
mode: -r-----
size: 769
num of links: 0

Inode Times:
Accessed:      Sat Nov 13 19:38:03 2004
File Modified: Sat Nov 13 19:37:21 2004
Inode Modified: Sat Nov 13 19:38:03 2004
Deleted:       Sat Nov 13 19:38:03 2004

Direct Blocks:
408298
```

For binary files, the *strings* tool was used. When strings is run given a filename as an input, it will output any content that has 4<sup>23</sup> or more consecutive ASCII characters, which in some instances can display words and sentences.

From the output obtained from *strings* being run against each file in turn, many of the executable files recovered appeared to be versions of the key binaries that were replaced by the rootkit. The output produced "help text" that is usually generated when the help or incorrect command line parameters are passed. By picking up on certain keywords and recognising some of the parameters described can provide details as to what it was originally.

The following, for example, was produced from the *strings* output run against file 245895. When this is compared with the output of running *ls --help*, the two are similar.

```
Usage: %s [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuSUX nor --sort.
  -a, --all                do not hide entries starting with .
  -A, --almost-all        do not list implied . and ..
  -b, --escape             print octal escapes for nongraphic
characters
  --block-size=SIZE        use SIZE-byte blocks
  -B, --ignore-backups     do not list implied entries ending with ~
  -c                       with -lt: sort by, and show, ctime (time of
last                               modification of file status
information)
```

---

<sup>23</sup> 4 is the default value, and may be changed by passing further commandline parameters.

-C

with -l: show ctime and sort by name  
otherwise: sort by ctime  
list entries by columns

What is not clear is whether the binary files recovered are the originals that were installed with the operating system, or whether these are Trojan copies.

<http://www.knowngoods.org> hosts a database containing the md5 and sha1 hashes of original files. By comparing the md5 hashes of the recovered files against this database it proved that they are original unaltered copies. Nothing of any significance was found. A summary of the files recovered can be found below.

199509	data	Empty
199513	data	Empty
199514	ASCII	/etc/passwd file
199515	ASCII	/etc/shadow file
245895	ELF	File: /bin/ls MD5: 5ec59b9c05706b4ce65adf44d0d3ab24 SHA-1: b1ce95da83b29dabd01ffbf695542cbf12e8df77 Size: 43024 (bytes) Platform: Linux RedHat 6.2 (i386)
245905	ELF	File: /usr/bin/du MD5: bf0627ce5e90e322d4e32982d231df64 SHA-1: 2354c4afc95f14990521bd0649e949923aa42c47 Size: 24752 (bytes) Platform: Linux RedHat 6.2 (i386)
246047	ELF	File: /bin/ps MD5: 5e1725f2734365fef9e55398785f3033 SHA-1: 024d5411eae8569404ccbf9f77bee155ad06869 Size: 60080 (bytes) Platform: Linux RedHat 6.2 (i386)
246051	ELF	File: /usr/bin/top MD5: 48fbbb48204825866ab3089c2db96e87 SHA-1: d4cd109d90139174304d8c44a1e9039803019e02 Size: 34896 (bytes) Platform: Linux RedHat 6.2 (i386)
247381	ELF	File: /bin/netstat MD5: f174e862d00d0998c3fa4ccd632019b5 SHA-1: fecf1eb33e7ed16e5a215093287efc586960a04 Size: 66736 (bytes) Platform: Linux RedHat 6.2 (i386)
247553	ELF	File: /usr/bin/rlogin MD5: 50c83d1390414f16064d3d9bf238bf0c SHA-1: ac9a97085a783e06e43d09485f8d71407b9493bf Size: 10256 (bytes) Platform: Linux RedHat 6.2 (i386)
247843	ELF	File: /bin/login MD5: 9b34aed9ead767d9e9b84f80d7454fc0



		SHA-1: 60bd34f7abc3cfda427c43888d1118d65cfc2165 Size: 20452 (bytes) Platform: Linux RedHat 6.2 (i386)
~etc~shadow.lock	data	Empty
~home~httpd~html~index.html~	data	Empty
~home~httpd~html~.index.html.swp	data	Backup file for /home/httpd/html/index.html while being edited by vi
~var~lock~makewhatis.lock	data	Lock file
~var~spool~mqueue~xfUAA16390	ASCII	Email message. Contents not relevant to case
~var~tmp~rpm-tmp.88454	ASCII	Unknown. No useful information.

## ***strings Analysis***

*strings* as has been seen previously can be very useful when searching for clues and evidence. When combined with *grep*, searches for specific words can be performed, which can reveal previously hidden information.

It does have its limitations though. *strings* will not find words within compressed or encrypted files. It will also not find any words that have other non-ASCII characters separating any of the letters, for example a null.

The process of searching a disk image using *strings*, and then finding the original file is as follows. For this example a search will be performed for the word "ch0pper" as this is known to exist.

The first stage is to trawl the image and locate any instances of the word ch0pper. This is done in the first instance using the *strings* command giving the image that is to be searched as a parameter. The *--radix=d* switch will ensure that the byte offset in decimal within the file will also be included in the output. As *strings* will output everything that contains 4 or more consecutive ASCII characters, the output needs to be filtered so it is therefore then piped through *grep*.

*grep* takes the output from *strings* and parses it for the occurrence of "ch0pper". The *-i* flag tells *grep* to ignore case and treat upper and lower case letters the same. The filtered output is then written into the file ch0pper.strings.

```
[root@localhost fi1001]# strings --radix=d ../image/fi1001.sda1.dd |grep -i ch0pper > ch0pper.strings
```

Upon examination of the ch0pper.strings file once the operation is complete, a number of occurrences can be found.

```
[root@localhost fi1001]# more ch0pper.strings
706100850 ++++0wN3d bY cH0PpeR+++++
706126199 ++++0wN3d bY cH0PpeR+++++
706141807 ++++0wN3d bY cH0PpeR+++++
1672134656 root:x:0:root,ch0pper
1672135168 ch0pper:x:501:
1672142848 root:::root,ch0pper
1672143275 ch0pper!:
1672151916 ch0pper:x:501:501::/home/ch0pper:/bin/bash
1672217452 ch0pper:x:501:501::/home/ch0pper:/bin/bash
1672381292 ch0pper:x:501:501::/home/ch0pper:/bin/bash
```

These entries now need to be seen in context and be mapped to a physical file on the filesystem. Before this can be done, however manual calculations need to be performed and the fragment size used needs to be known.

The *fsstat* tool can be used to obtain this information.

```
[root@localhost fi1001]# fsstat -f linux-ext2 ../image/fi1001.sda1.dd |more
```

#### FILE SYSTEM INFORMATION

```
-----  
File System Type: EXT2FS  
Volume Name:  
Last Mount: Thu Nov  4 05:08:17 2004  
Last Write: Sat Nov 13 22:26:13 2004  
Last Check: Wed Nov  3 14:24:15 2004  
Unmounted Improperly  
Last mounted on:  
Operating System: Linux  
Dynamic Structure  
InCompat Features: Filetype,  
Read Only Compat Features: Sparse Super,
```

#### META-DATA INFORMATION

```
-----  
Inode Range: 1 - 507904  
Root Directory: 2
```

#### CONTENT-DATA INFORMATION

```
-----  
Fragment Range: 0 - 1014094  
Block Size: 4096  
Fragment Size: 4096
```

Once the fragment size is know, the fragment number needs to be calculated.  
This is done using the following formula.

$$\text{fragment} = (\text{decimal offset in image file}) \div (\text{fragment size})$$

The occurrence that looks like it is part of the /etc/group file will be used, the calculation is performed

```
1672134656 root:x:0:root,ch0pper  
408236      = 1672134656 ÷ 4096
```

This gives the fragment number.

The ifind tool is then used to locate which inode points to this fragment.

```
[root@localhost fi1001]# ifind -f linux-ext2 ../image/fi1001.sda1.dd -d  
408236  
198599
```

ffind is then used to provide a filename using the inode number produced by ifind as input.

```
[root@localhost fi1001]# ffind -f linux-ext2 ../image/fi1001.sda1.dd  
198599  
/etc/group
```

Which does indeed lead us back to the /etc/group file.

This method was used to search for the following keywords within all images, including sda5 which was the swap partition (case insensitive):

ch0pper	Searching for "ch0pper" may highlight other files that have been modified.
81.146	The IP addresses used by the attacker always had 81.146 as the first two octets. A search may reveal other log files that not yet been analysed.
172.26.1.	If a packet sniffer is running and logging, it would have most certainly seen traffic on the local subnet so therefore IP addresses with these first three octets would have been recorded in a file. That is assuming that the log file is in plain text and is not compressed or encrypted in any way.

None of the above searches revealed anything of interest above and beyond what has already been discovered. The absence of any interesting results from the 172.26.1 search has indicated that while a packet sniffer was installed as part of the rootkit, it appears that it was never run.

## ***Conclusion and Timeline of Activities***

On Saturday November 13<sup>th</sup> at 21:48:57 the default web page was defaced. In the defacement the attacker left details of his/her hacking handle, ch0pper, and also thanked us for giving him/her wu-ftpd.

The version of wu-ftpd running on the system was 2.6.0. The vulnerability database at Security Focus<sup>24</sup> lists a number of issues with this particular version, in particular a number<sup>25</sup> that could allow for remote execution of arbitrary code on the system and therefore present an avenue to gain unauthorised entry.

This being the case and the fact that the defacement mentioned the wu-ftp service, it can be assumed that this was the route in although there is no evidence to show that this was indeed the method used.

Several anonymous logins were successfully made to the ftp server, but upon analysing the ftp logs, they only show two events - the uploading of two tar archives that appear to be rootkits.

Once the attacker had gained access to the system, a user account called ch0pper was created and added to the root group. Surprisingly, the attacker did not give the ch0pper user a uid of 0 which would have given it the same privileges as root.

Version 2 of the t0rnkit was found to be installed, but no evidence of the sniffer being used could be found. Additionally, it appears the attacker also tried to install the lrk5 rootkit as well, but unsuccessfully.

The final actions of the attacker were to deface the website itself. As the network connection for the server was pulled shortly after this no further activity was recorded.

From the activity observed, it is unclear what the intentions of this attacker were. The attacker created a normal user account called ch0pper, which also in turn created a home directory. The presence of a user account such as this, and also the home directory containing two rootkits is likely to be noticed by a system administrator in a short amount of time if it is regularly used. The default web page was also defaced, which is also a good way to alert people to the presence of a hacker being on the system.

The purpose of a rootkit is to hide any suspicious activity from system administrators and allow an attacker to gain subsequent entry and use a system unnoticed. Being that the attacker has been so noisy, what was the point of installing a rootkit? More to the point, what was the point of attempting to install two rootkits? After all, the first one appears to have installed successfully. The fumbling around trying to find the default web page suggests that the attacker was not very familiar with old linux systems. The failure to give the ch0pper user root level access once created also shows some immaturity of

---

<sup>24</sup> Security focus hosts a database which holds details about known vulnerabilities within many systems. <http://www.securityfocus.com/bid>

<sup>25</sup> Vulnerabilities listed for version 2.6.0 of wu-ftpd within the Security Focus vulnerability database<sup>24</sup> that could allow for remote arbitrary code execution have the following BID (Bugtraq ID) numbers: 1387, 3581, 2240, 8668, and 8315.

knowledge. Since there was no packet sniffer found to be running and the accesses of the README files, it would also suggest that the attacker did not really understand the tools that he/she had installed.

From the activity observed, it appears that ch0pper may be fairly young in experience when it comes to hacking. He/she saw an opportunity to compromise a host, did so, and once entry had been gained was unsure as to what to do, nor how to cover their tracks.

Below is a timeline of activity as generated by the evidence gained.

10<sup>th</sup> Nov 2004 04:16 Failed telnet login from 81.146.25.222  
13<sup>th</sup> Nov 2004 07:12 Brute force attack from 81.146.41.184. Not successful.  
07:25 Manual http connection initiated from 81.146.41.184  
07:34 Automated http scan from 81.146.41.184 for vulnerable scripts or files that could provide useful information  
18:41 Failed ftp root login from 81.146.41.184  
18:50 Failed ftp login for unknown user from 81.164.41.184  
18:51 Successful anonymous ftp login from 81.164.41.184. fred@fred.com passed as password. Further successful logins are observed until 19:28  
19:37 User ch0pper is created  
19:38 User ch0pper logs into system from 81.146.41.184  
19:53 Attacker views the Apache configuration file  
20:21 User ch0pper logs into system from 81.146.45.181  
20:43 t0rnkit.tar is uploaded via ftp  
20:49 The ch0pper.sh script that provides an interactive bash shell is created on the system. It is never used however.  
20:52 t0rnkit.tar is extracted  
20:56 lrk5.src.tar.tar is uploaded via ftp  
20:56 t0rn rootkit is installed  
21:37 lrk5.src.tar.tar archive is extracted.  
Configuration script in /home/ch0pper/lrk5 is run.  
21:39 Make was run, but failed. Most likely in an attempt to make the lrk5 rootkit  
21:41 README file for t0rn rootkit is viewed.  
21:48 Default web page is defaced  
21:50 A netcat listener is attempted to be created.  
22:14 Locate is run to find in.amqd  
22:30 Power cord removed from server.

## References

RedHat, Inc. "RedHat Linux". URL: <http://www.redhat.com> (22<sup>nd</sup> November 2004)

Hewlett Packard. "Compaq". URL: <http://www.compaq.com> (22<sup>nd</sup> November 2004)

Carrier, Brain. "The Sleuthkit". 1.72. September 7<sup>th</sup> 2004.  
URL: <http://www.sleuthkit.org/sleuthkit/> ( 22<sup>nd</sup> November 2004)

Microsoft Corporation. "Microsoft Windows". URL: <http://www.microsoft.com> (22<sup>nd</sup> November 2004)

VMWare, Inc. "VMWare" URL: <http://www.vmware.com> (22<sup>nd</sup> November 2004)

The Apache Software Foundation. "Apache". URL: <http://www.apache.org> (22<sup>nd</sup> November 2004)

"Knoppix Security Tools Distribution". URL: <http://www.knoppix-std.org> (22<sup>nd</sup> November 2004)

"Knoppix". URL: <http://www.knoppix.org> (22<sup>nd</sup> November 2004)

@stake, Inc. "The @stake Sleuth Kit". URL: <http://www.atstake.com> (22<sup>nd</sup> November 2004)

Google. "Google" URL: <http://www.google.com> (22<sup>nd</sup> November 2004)

RIPE. "Ripe Whois Database". URL: <http://www.ripe.net> (22<sup>nd</sup> November 2004)

The Shmoo Group. "Known Goods". URL: <http://www.knowngoods.org> (22<sup>nd</sup> November 2004)

Security Focus. "Security Focus vulnerability database". URL: <http://www.securityfocus.com/bid> (22<sup>nd</sup> November 2004)