

File System Analysis Techniques

Sleuth Kit Reference Document

www.sleuthkit.org

Brian Carrier

Last Updated: July 2005

Introduction

Currently, evidence is most frequently found in the file system. This is because it is non-volatile and remnants of deleted files can typically be found. This file will help one to use the low-level tools in The Sleuth Kit for a forensic analysis.

This document is organized into small scenarios, which provide examples of how to use The Sleuth Kit. Most of these functions are automated with Autopsy, but they are here for reference and education.

<http://www.sleuthkit.org/autopsy>

The techniques used here apply to both UNIX and Windows file systems.

Timelines

The steps from the timeline Sleuth Kit Implementation Notes are followed (using both `ils` and `fls`) and you notice some interesting activity from unallocated inodes, namely MFT Entry 5035 from image `c_drive.dd`. To display the contents of this file, use "icat":

```
# icat images/c_drive.dd 5035 | less
```

NOTE: To prevent your terminal from getting messed up, pipe all output of "icat" through a pager like "less".

Search

In this scenario, we will search the unallocated space of the "wd0e.dd" image for the string "abcdefg". The first step is to extract the unallocated disk units using the "dls" tool (as this is an FFS image, the addressable units are fragments).

```
# dls images/wd0e.dd > output/wd0e.dls
```

Next, use the UNIX strings(1) utility to extract all of the ASCII strings in the file of unallocated data. If we are only going to be searching for one string, we may not need to do this. If we are going to be searching for many strings, then this is faster. Use the '-t d' flags with "strings" to print the byte offset that the string was found.

```
# strings -t d output/wd0e.dls > output/wd0e.dls.str
```

Use the UNIX grep(1) utility to search the strings file.

```
# grep "abcdefg" output/wd0e.dls.str | less
10389739: abcdefg
```

We notice that the string is located at byte 10389739. Next, determine what fragment. To do this, we use the 'fsstat' tool:

```
# fsstat images/wd0e.dd
<...>
CONTENT-DATA INFORMATION
-----
Fragment Range: 0 - 266079
Block Size: 8192
Fragment Size: 1024
```

This shows us that each fragment is 1024 bytes long. Using a calculator, we find that byte 10389739 divided by 1024 is 10146 (and change). This means that the string "abcdefg" is located in fragment 10146 of the "dls" generated file. This does not really help us because the dls image is not a real file system. To view the full fragment from the dls image, we can use dd:

```
# dd if=images/wd0e.dd bs=1024 skip=10146 count=1 | less
```

Next, we will identify where this fragment is in the original image. The "dcalc" tool will be used for this. "dcalc" will return the "address" in the original image when given the "address" in the dls generated image. (NOTE, this is currently kind of slow). The '-u' flag shows that we are giving it an dls address. If

the '-d' flag is given, then we are giving it a dd address and it will identify the dls address.

```
# dcalc -u 10146 images/wd0e.dd
59382
```

Therefore, the string "abcdefg" is located in fragment 59382. To view the contents of this fragment, we can use "dcat".

```
# dcat images/wd0e.dd 59382 | less
```

To make more sense of this, let us identify if there is a meta data structure that still has a pointer to this fragment. This is achieved using "ifind". The '-a' argument means to find all occurrences.

```
# ifind -a images/wd0e.dd 59382
493
```

Inode 493 has a pointer to fragment 59382. Let us get more information about inode 493, using "istat".

```
# istat images/wd0e.dd 493
inode: 493
Not Allocated
uid / gid: 1000 / 1000
mode: rw-----
size: 92
num of links: 1
Modified: 08.10.2001 17:09:49 (GMT+0)
Accessed: 08.10.2001 17:09:58 (GMT+0)
Changed: 08.10.2001 17:09:49 (GMT+0)
Direct Blocks:
59382
```

Next, let us find out if there is a file that is still associated with this (unallocated) inode. This is done using "ffind".

```
# ffind -a images/wd0e.dd 493
* /dev/.123456
```

The leading '*' identifies the file as deleted. Therefore, at one point, the file '/dev/.123456' allocated inode 493, which allocated fragment 59382, which contained the string "abcdefg".

If "ffind" returned with more than file that had allocated inode 493, it means that either both were hard-links to the same file or that one file (chicken) allocated the inode, it was deleted, a second file (egg)

allocated it, and then it was deleted. The string belongs to the second file, but it is difficult to determine which came first. On the other hand, if "ffind" returns with two entries where one deleted and one not, then the string belongs to the non-deleted file.

As previously mentioned, Autopsy will do all of this for you when you do a keyword search of unallocated space.

Deleted Content

To view all of the deleted file names in an image, use the "fls" tool. For all deleted files, use the '-r' flag for recursive and '-d' flag for deleted.

```
# fls -rd images/hda9.dd | less
d/d * 232: /TEMP-823450
r/d * 293: /TEMP-131100
```

This shows us the full path that the deleted files are located. On some systems, such as Windows NTFS, the file content may be recovered (depending on how much system activity has occurred). On other systems, such as Solaris UFS and Linux Ext3, deleted files can not be easily recovered. The number at the beginning of the line is the inode number. The '*' shows that it is deleted and the 'd' and 'r' show the type (directory and file). The first letter identifies the directory entry type value (which does not exist in all file system types) and the second letter is the type according to the inode. In most cases these should be the same, but it may not for deleted files if the inode has been reallocated to a file of a different type. If we do an "istat" on the directory (232) we will notice that the size is 0.

```
# istat images/hda9.dd 232
inode: 232
Not Allocated
uid / gid: 0 / 0
mode: rwxr-xr-x
size: 0
num of links: 0
Modified: 08.23.2001 21:52:33 (GMT+0)
Accessed: 08.23.2001 23:05:39 (GMT+0)
Changed: 08.23.2001 21:52:33 (GMT+0)
Deleted: 08.23.2001 23:05:39 (GMT+0)
Direct Blocks:
```

Linux does this to all of its deleted directories. It should also be observed that no block addresses are shown in the "istat" output. This is because the size is 0 and the program thinks that the address is bogus.

Using the '-b' option of "istat", we can force it to output the block address. With Linux Ext3, the block pointers would be 0, but Linux Ext2 kept the old addresses.

```
# istat -b 2 images/hda9.dd 232
inode: 232
Not Allocated
uid / gid: 0 / 0
mode: rwxr-xr-x
size: 0
num of links: 0
Modified: 08.23.2001 21:52:33 (GMT+0)
Accessed: 08.23.2001 23:05:39 (GMT+0)
Changed: 08.23.2001 21:52:33 (GMT+0)
Deleted: 08.23.2001 23:05:39 (GMT+0)
Direct Blocks:
388 0
```

Now we can examine the contents of block 388 and see the file names that were in that directory:

```
# dcat -h images/hda9.dd 388 | less
```

Manual UNIX File Recovery

A UFS/FFS or EXT2FS/EXT3FS file system is organized into groups. Each group has its own inodes and blocks to store data in. When a new file is created, it is given an inode in the same group that the parent directory inode is in (if there are still inodes available). When a new directory is created, it is given an inode in a new group. An inode allocates blocks from the same group that its inode is in.

When recovering a file from one UFS or EXTxFs, the group layout can be used. When a deleted file is found with 'fls', notice the inode of the parent directory:

```
# fls -r images/hda1.dd
d/d 30789: doc
+ r/r * 0: doc/.a/ssh.tar
+ r/r 30792: doc/.a/install
<...>
```

We want to recover the 'ssh.tar' file and notice that the parent directory is 30789 and the deleted file has a cleared inode pointer. To identify the group that it is in, the 'fsstat' tool is used:

```
# fsstat images/hda1.dd
FILE SYSTEM INFORMATION
-----
File System Type: EXT3FS
<...>

Group: 0:
  Inode Range: 1 - 15392
  Block Range: 0 - 32767
  Super Block: 0 - 0
  Group Descriptor Table: 1 - 1
  Data bitmap: 2 - 2
  Inode bitmap: 3 - 3
  Inode Table: 4 - 484
  Data Blocks: 485 - 32767

Group: 1:
  Inode Range: 15393 - 30784
  Block Range: 32768 - 65535
  Super Block: 32768 - 32768
  Group Descriptor Table: 32769 - 32769
  Data bitmap: 32770 - 32770
  Inode bitmap: 32771 - 32771
  Inode Table: 32772 - 33252
  Data Blocks: 33253 - 65535

Group: 2:
  Inode Range: 30785 - 46176
  Block Range: 65536 - 98303
  Data bitmap: 65536 - 65536
  Inode bitmap: 65537 - 65537
  Inode Table: 65540 - 66020
  Data Blocks: 65538 - 65539, 66021 - 98303
<...>
```

The inode is in the range of inode addresses for group 1. To search for the deleted file, we extract the unallocated space using 'dls':

```
# dls images/hda1.dd 32768-65535 > output/hda1-grp1.dls
```

If we wanted to extract all of the data for the group, we could use 'dd':

```
# dd if=images/hda1.dd of=output/hda1-grp1.dd bs=4096 skip=32768
```

```
count=32767
```

Where, the fragment size is 4096 (which can also be found in the 'fsstat' output). Either of these images can then be analyze for keywords or using other data carving tools such as 'foremost'. This process allows one to reduce the amount of data that must be analyzed.

<http://foremost.sourceforge.net>

Copyright © 2002-2005 by Brian Carrier. All Rights Reserved