Performing An Autopsy Examination on FFS and EXT2FS Partition Images

An Introduction to TCTUTILs and the Autopsy Forensic Browser

Brian Carrier
[carrier@cerias.purdue.edu]

1 Introduction

The forensic analysis of a compromised hard drive has two major phases, the collection of data from the partitions and the analysis of the collected data. The collection of data from common UNIX drive formats such as Berkeley Fast File System (FFS) and Linux EXT2FS is well known. A bit-wise copy of the partition must be made to save all pieces of evidence and to easily verify that the copy is the same as the original. The most common way to do this is using the UNIX `dd(1)` command, and it was actually shown in SC Magazine that `dd` is the most reliable imaging software [6]. This paper will focus on the second phase, data analysis.

In this paper, a collection of free, open source forensics tools will be used: The Coroners Toolkit (TCT), TCTUTILs, and the Autopsy Forensic Browser (AFB). TCT was written by Wietse Venema and Dan Farmer and first released in August 2000 and the two latter software packages were written by the author and released in March 2001. TCTUTILs provides additional tools that are based on TCT and Autopsy is a graphical interface to TCT, TCTUTILs, and standard UNIX utilities. Both TCT and TCTUTILs read FFS and EXT2FS images created by `dd`, or a similar program that does not insert additional data such as CRC or hash values.

The paper begins with a brief description of how TCT can help in an investigation, introduce TCTUTILs, describe the features of Autopsy, and conclude with a set of examples. Throughout this document, all examples will use images from the Honeynet Forensic Challenge [3] so the reader can easily recreate them. The tools and techniques introduced in this paper were used in the 2nd place entry in the challenge. This is a technical paper and assumes that the reader has a basic understanding of FFS and EXT2FS file system details, such as inodes and blocks.

2 The Coroners Toolkit (TCT)

The Coroners Toolkit [5] (TCT) is an open source forensics package that can assist in the analysis of FFS and EXT2FS partition images. It includes a powerful C file system library, *fs_lib.a*, and several tools that can be used on mounted partitions and forensic images. This section describes some of the tools that maybe used with TCTUTILs and Autopsy, but is not intended as a thorough discussion of TCT.

2.1 grave-robber and mactime

The grave-robber tool is a data collection program that analyzes mounted images, both live and post-mortem. Every file in a FFS or EXT2FS file system has a Modified, Accessed, and Changed (MAC) time. When given the '–m' flag, grave-robber saves the MAC times of every file and directory into a file, named body. The tool can also save data such as deleted files that are still open and memory contents.

The mactime tool is a perl script that processes the body file. It creates an ASCII timeline, where each entry corresponds to the modification, access, or change of a file. A timeline is useful when trying to determine which files have been recently created or accessed. Furthermore, because many systems have large amounts of drive space, a timeline can show directories where an investigator should focus on. Of course, an attacker can easily change the MAC times and if too much time passes, the data related to the attack may be overwritten.

2.2 Ils and ils2mac

The ils tool displays data about unallocated inodes. It displays data such as file size and MAC times. The ils2mac script, converts the output of ils into the format of the body file from grave-robber. Therefore, one can concatenate the output of ils2mac into a body file and generate a timeline with entries for allocated and unallocated inodes. This will provide insight about when and where files were deleted.

2.3 unrm and lazarus

The unrm tool is a variant of dd and produces a stream of block contents. By default, it extracts the unallocated blocks from a partition image.

According to the man page, the goal of lazarus is to "create structure from unstructured data". It takes a stream of bytes as input and analyzes them in block size chunks. It tries to identify what type of data the block contains (i.e. C source, tar file, email) and makes a file that lists the guessed content type and a directory of files that contain block contents. Optionally, the output of lazarus is in HTML and an HTML browser can be used to examine the contents of each block. When used with unrm, unallocated blocks can be browsed to find deleted content. This process can be tedious for large partitions with lots of free space.

2.4 icat

The icat utility displays the contents of a file or directory that is specified by an inode and image. This is similar to using the UNIX cat command, but instead of using the file name as an argument, the inode is used. Therefore, this can be used to display the contents of some unallocated inodes (if the system does not delete the block pointers).

3 TCTUTILs

While TCT provides many powerful tools, there is missing functionality. Namely, the ability to list file and directory names in an image, the ability to map between the blocks and inodes, the ability to map between inodes and file names, and the ability to obtain details of a specific inode. For this reason, TCTUTILs was developed [2]. TCTUTILs provides additional tools that use the *fs_lib.a* library from TCT and also provides a new type of data structure and associated routines.

This section will first describe the new structures that were added and then describe the tools that are included in the 1.01 release of the package.

3.1 Technical Details

The TCTUTILS package includes a new data structure called FS_DENT, for file system directory entry. The purpose of the FS_DENT structure is to integrate file and directory names into the forensic analysis when using TCT.

In FFS and EXT2FS, every directory has an associated inode and allocated blocks. For example, the root directory of a partition is always inode 2 and for this example it has allocated block 258. The contents of block 258 are a series of structures of type *struct dirent* in FFS and *struct ext2_dir_entry_2* in EXT2FS. There are minor differences between the structures, but that is not important here. The fields of interest are:

- inode: inode number for file with this name
- rec_len: length in bytes of this structure (rounded up to a multiple of 4)
- nam_len: length in bytes of the file name
- name: actual file name in ASCII

To list the contents of a directory, one must determine which blocks the directory has allocated. Next, one of the previously mentioned structures is used as a template, starting at byte 0 of the block. The structure will point to where the entry name begins and where the *rec_len* field is. To find the next entry in the directory, the template is moved ahead *rec_len* bytes, where it will point to the next name and associated fields.

When a file is deleted, for performance reasons, the directory entry is not removed. The *rec_len* field of the previous entry is incremented by the length of the deleted entry. When the steps described in the previous paragraph for listing the directory contents are performed, the deleted file will be skipped. For example, Table 1 shows the field values for two consecutive directory entries. If *file2* is deleted, the result is shown in Table 2.

Table 1 Fields of two active directory entries

inode	**rec_len**	**nam_len**	**name**
8357	16	5	file1
8358	16	5	file2

Table 2 Fields of one deleted and one active directory entries

inode	**rec_len**	**nam_len**	**name**
8357	<u>32</u>	5	file1
8358	16	5	file2

Therefore, one can easily identify which files have been recently deleted because their length is much longer then it should be, based on the file name length. A simple check is if *rec_len* is greater than the actual length of the current record plus the length of a record with a file name of one character. By searching through the directory blocks, one is able to identify which files have recently been deleted.

The amount of information that can be gathered by this method is OS dependent. There are two types of data that are typically deleted when a file is deleted: the inode value in the directory entry and inode contents such as file size and block pointers. If the inode value in the directory entry is deleted, then only the file name can be determined. This will still be useful when searching in unallocated blocks. If the file size and block pointers are deleted, the Modified, Access, and Change times of the file can be determined. If nothing is deleted, one can recover all or part of a file. Table 3 shows which of these actions Debian Linux 2.2, OpenBSD 2.8, and Solaris 2.7 perform. Linux does not delete any fields and therefore recently deleted files can be easily recovered. OpenBSD deletes the block pointers and file size, so MAC times can be determined. Solaris deletes everything, so only file names can be recovered.

Table 3 Status of fields after a file is deleted

	Inode field in directory entry	**File size and block pointers**
Debian Linux 2.2	Saved	Saved
OpenBSD 2.8	Saved	Deleted
Solaris 2.7	Deleted	Deleted

3.2 Tools

This section will describe the tools that are included in TCTUTILs v1.01. They are presented in an order that allows for examples to be easily presented.

3.2.1 istat

The istat tool displays all known data about an inode, including all of the blocks that it has listed in its block pointers and the addresses of the indirect blocks. For example, running this on the forensics challenge root directory file shows:

```
# istat honeypot.hda8.dd 2
inode: 2
uid / gid: 0 / 0
mode: rwxr-xr-x
size: 1024
num of links: 19
Modified:     11.08.2000 09:52:25     (EST+0)
Accessed:     11.08.2000 05:02:02     (EST+0)
Changed:      11.08.2000 09:52:25     (EST+0)
Deleted:      12.31.1969 19:00:00     (EST+0)
Direct Blocks:
  258
```

This is useful for generating reports on inodes and provides a nicer format than using ils in TCT. Also, ils will not display all of the blocks that the inode uses. The indirect blocks were not displayed in the 1.0 version of this tool.

3.2.2 bcat

The bcat tool is a simple way to display the contents of an image block. It can display the contents in raw, ASCII, or hexdump format and can also display them as an HTML table. This tool uses the TCT *fs_lib.a* library, but the same functionality could be achieved using dd and passing it to a formatting program. When bcat is run on the block of the root directory of honeynet.hda8.dd using the hexdump output, one gets:

```
# bcat -h honeypot.hda8.dd 258 512
0        02000000 0c000102 2e000000 02000000   .... .... .... ....
16       0c000202 2e2e0000 0b000000 14000a02   .... .... .... ....
32       6c6f7374 2b666f75 6e640000 c10f0000   lost +fou nd.. ....
48       0c000402 626f6f74 811f0000 0c000402   .... boot .... ....
64       686f6d65 412f0000 0c000302 75737200   home A/.. .... usr.
80       013f0000 0c000302 76617200 c14e0000   .?.. .... var. .N..
96       0c000402 70726f63 a1560000 0c000302   .... proc .V.. ....
112      746d7000 815e0000 0c000302 64657600   tmp. .^.. .... dev.
128      61660000 0c000302 65746300 21760000   af.. .... etc. !v..
144      0c000302 62696e00 e1850000 0c000302   .... bin. .... ....
160      6c696200 a1950000 0c000302 6d6e7400   lib. .... .... mnt.
176      41ad0000 0c000302 6f707400 21b50000   A... .... opt. !...
192      0c000402 726f6f74 01bd0000 0c000402   .... root .... ....
208      7362696e 52ec0000 10000602 666c6f70   sbin R... .... flop
224      70790000 17000000 18000d07 2e626173   py.. .... .... .bas
240      685f6869 73746f72 79000000 fe070000   h_hi stor y... ....
256      04030a02 2e617574 6f6d6f75 6e740000   .... .aut omou nt..
[...]
```

The directory and file names can be seen in the ASCII representation on the right and the inodes and record lengths can be seen on the left. bcat can also display the contents of a swap page with the same format options.

3.2.3 find_inode

The find_inode tool searches an image for an inode that has a given block in its list of block pointers. There are three possible scenarios when this is run: the block will be in an inode’s list, the block will not be in an inode’s list, or the block is contained within a larger fragment. For example:

```
# find_inode honeypot.hda8.dd 257
No inode or potential fragment base was found
# find_inode honeypot.hda8.dd 258
2
```

Using OpenBSD a fragment example can be observed:

```
# istat wd0e 4
inode: 4
uid / gid: 0 / 0
mode: rw-r--r--
size: 2164
num of links: 1
Modified:      01.21.2001 16:12:25     (EST+0)
Accessed:      01.21.2001 16:43:42     (EST+0)
```

```
Changed:       01.22.2001 15:47:12     (EST+0)
Direct Blocks:
  262
```

Notice the file size is larger than the 1024 byte block size, but only one block is allocated.

```
# find_inode wd0e 262
4
# find_inode wd0e 263
Block 263 could be part of file block 262 for inode 4
```

As of version 1.01, find_inode will also identify an inode that is using a block as an indirect block to save the addresses of direct or additional indirect blocks. This tool is useful when searching an image for strings or keywords. If the block that a keyword appears in has been identified, this tool will quickly identify if it is part of a larger file that should be looked at and possibly recovered using icat (TCT).

3.2.4 fls

The fls utility is the largest tool to use the new FS_DENT structures. The fls tool lists the files and directories that have directory entries in a directory allocated block. The tool will print deleted entries with a '*' before them and is able to traverse directory entries. Due to space limitations, a small example will be created instead of a large directory from the Honeynet Project:

```
# echo "File 1"> file1
# mkdir dir1
# echo "File 2" > file2
# echo "File 3" > file3
# echo "File 1 in dir 1" > dir1/file1
# ls -R
dir1/  file1  file2  file3
./dir1:
file1
# fls -r /dev/rwd0e 16
r 19:    file1
d 1797:  dir1
+ r 1808:     file1
r 22:    file2
r 23:    file3
# rm file2
# fls -r /dev/rwd0e 16
r 19:    file1
d 1797:  dir1
+ r 1808:      file1
r * 22:  file2
r 23:    file3
```

The first character shows what type of files it is (i.e. regular, directory, link, socket) and the first number is the inode number. The '+' sign shows how deep the file is in the directory tree. Using other arguments to fls, one can list only the deleted files, only the undeleted files, only directories, only files, or print a long version that gives MAC times, sizes, UID, and GID.

This tool has many obvious uses. First, it can list all of the deleted files to get a better understanding of what tools the attacker installed on the system. The second use is that it allows one to view the contents of a partition without having to mount it in loop back, which is useful when that option is not available.

Another option with fls is to print data in time machine (TCT) format, using the '–m' option. This will print out file statistics for deleted files on systems that do not delete the inode value in the directory entry. The time machine format is the same output as from grave-robber (TCT), so the two outputs can be combined and processed by mactime (TCT). Therefore, using this data one can make a more accurate timeline of file events when starting an investigation. An example of this is given later.

3.2.5 find_file

The find_file tool also uses the FS_DENT structures and finds the file(s) that have a specified inode in their directory entry. It recursively walks through the directories starting at the root directory (inode 2) and looks for an entry that has the requested inode. On systems such as OpenBSD and Linux where the inode field is not deleted, it can also identify recently deleted files. For example, to find the files and directories that use inode 24193 one would do:

```
# find_file -a honeypot.hda8.dd 24193
/dev
/dev/.
/dev/ida/..
/dev/pts/..
/dev/raw/..
/dev/rd/..
```

This example also shows that multiple files or directories can have links to one inode. By using the '–a' switch, find_file will find all occurrences. To determine which file allocated inode 109791 on hda5:

```
# find_file -a honeypot.hda5.dd 109791
* /man/.Ci/ssh-1.2.27.tar
```

It can be seen that this inode is from a deleted tar file.

3.2.6 blockcalc

The blockcalc tool converts between block numbers from an unrm generated image and block numbers from the original image. The lazarus (TCT) program is often run on an image that was generated with unrm (TCT). An unrm-generated image contains a subset of the total blocks in the original image and there is not an obvious mapping to the original. This becomes an issue while documenting data found using lazarus and when using both lazarus and autopsy. The blockcalc tool creates a map between the two images and can convert the block number from one image to the block number in the other.

4 Autopsy Forensic Browser

The Autopsy Forensic Browser (AFB) [1] is a graphical front-end to TCT, TCTUTILs and standard UNIX utilities. It contains no additional tools except the 'glue' to integrate the tools from TCT and TCTUTILs into an easy to use interface. Autopsy is HTML-based and requires an HTML browser that supports frames and forms. The autopsy program is written in perl and acts as an HTML server. It only allows connections from one host specified at execution time and uses a random cookie for basic authentication.

The browser was designed such that it contains no analysis tools. Its purpose is to integrate many command line tools for procedures that are commonly performed. Yet, for circumstances when Autopsy does not provide the needed functionality, an investigator can use the individual command line tools. This creates an environment where routine procedures are made easier, but an investigator is not always constrained by a graphical interface. As all of the tools are open source, the investigator can verify that the tools and interface are doing the 'correct' thing.

The browser currently has 4 major functions:

- File and Directory Browsing
- Block Browsing
- Inode Browsing
- Block Searching

Example screen shots are given in Appendix A.

4.1 File and Directory Browsing

The file and directory browsing feature provides a user interface similar to a file manager. The screen is arranged into three frames. On the left-hand side, all directories of an image are listed. The right-hand side is broken up into a top and a bottom half. By selecting a directory name on the left-hand side, the files and directories contained in it are listed in the upper right-hand side. The details of the files are given, such as MAC times, inode number and size. Both the left-hand side and the upper right-hand side listings are generated with fls (TCTUTILs), so recently deleted file names will be shown. The file names and inode numbers of each entry in the upper right-hand side can be selected, with the following effects:

- File Name: The file contents are displayed in the lower half using icat (TCT). If a binary is being displayed, one can choose to run the output through the UNIX utility strings(1) first. On some systems, such as Linux, a recently deleted file can be viewed.
- Directory Name: The directory is opened in the upper right-hand window. Deleted directories cannot be opened.
- inode: The inode details are displayed in the lower half using istat (TCTUTILs). Each block that is listed is a link to display its contents in block browsing mode.

This format allows the investigator to easily browse the files and view their contents. No file is ever read in its native format. For example, when a JPEG graphic file is selected, then the stream of bytes will not be sent with an image context type. Therefore, the HTML browser will not interpret it as a picture. This is to protect the investigators system from hostile code, but may be inconvenient for cases involving a large number of pictures. Future versions may provide an option for this.

The strings(1) output allows for easy executable decompilation to find backdoor passwords or files that contain trojan configuration settings. An example of this will be given in Section 5.2.

4.2 Inode Browsing

The inode browsing mode displays the output of istat (TCTUTILS) for specified inodes. The investigator enters an inode number and the details of the inode are displayed. For each block that is allocated for the inode, an HTML link will display the block contents. The file names that access the inode will be listed as well, using find_file.

Inode browsing mode also allows one to easily view the next or previous inodes to scan all inode entries.

4.3 Block Browsing

The block browsing mode allows the investigator to easily view the contents of blocks in several formats; it is a front-end to bcat (TCTUTILs). The screen contains a menu on the left-hand side and the block contents are given on the right. The main screen contains a field where a block number can be entered and a pull-down menu. The menu allows an investigator to enter a block number from either a dd or unrm generated image. The program will convert the unrm block number to the original number. After the block number is entered into the appropriate field, the contents are displayed in ASCII format. Additional formats can be used, such as strings and hexdump. In addition to the block contents, it will also show which inode has a pointer to it, by using find_inode. If the inode is found, it will also print the file name, by using find_file.

The top of the window contains links to view the previous and next blocks so the investigator can easily examine the neighboring blocks. This allows one to search for deleted contents.

This function has some overlap with lazarus (TCT). lazarus tries to guess what type of file was in the block, which is a feature that Autopsy currently does not have. Therefore, it is useful to use both of them together. For example, if one is looking for tar files, then lazarus can identify where the blocks are and Autopsy can determine if that block is part of a larger file, whose inode is still intact. This was the motivation for the unrm block number input option.

4.4 Block Searching

The block-searching mode is an easy way to search for keywords. Currently, only strings can be searched for, but future versions will support regular expressions. This mode arranges the screen into two frames. The left side contains a list of blocks where the string was found and the right side displays the block contents, using block browsing mode. This provides a convenient format of searching for keywords without having to compute block offsets by hand

4.5 Autopsy Reports

Another convenient feature of Autopsy is report generation. Autopsy can generate an ASCII report for any file, block, or inode that is of interest. The report includes data such as MD5 hash values of the data, investigators name, report generation time, and other meta data. The data can be printed in the hexdump, strings, or ASCII formats. This allows recovered blocks and files to be presented in a common format.

5 An Example Autopsy Examination

This section provides examples on how to use TCT, TCTUTILs, and Autopsy using images from the Honeynet Forensic Challenge [3]. The environment used will be described first.

All work is done in a directory created for the challenge, /usr/local/forensics/chal. The images are located in the images subdirectory and mounted in loop back mode with read-only and no-execute permissions on a subdirectory named mnt. These commands are taken from the challenge rules [3].

```
# pwd
/usr/local/forensics/chal
# mkdir mnt
# mount -o ro,loop,nodev,noexec images/honeypot.hda8.dd mnt
# mount -o ro,loop,nodev,noexec images/honeypot.hda1.dd mnt/boot
# mount -o ro,loop,nodev,noexec images/honeypot.hda5.dd mnt/usr
# mount -o ro,loop,nodev,noexec images/honeypot.hda6.dd mnt/home
# mount -o ro,loop,nodev,noexec images/honeypot.hda7.dd mnt/var
```

The images directory is used as the morgue directory for autopsy. The fsmorgue file contains the following entries:

```
# cat images/fsmorgue
honeypot.hda1.dd     /boot/
honeypot.hda5.dd     /usr/
honeypot.hda6.dd     /home/
honeypot.hda7.dd     /var/
honeypot.hda8.dd     /
```

5.1 Timeline

5.1.1 Creation

Using TCT and TCTUTILs, a detailed timeline of file activity can be created. grave-robber (TCT) is run first to save the Modified, Accessed, and Changed (MAC) times of the images mounted under the mnt directory. All data is saved to the data directory.

```
# mkdir data
# grave-robber -c mnt -m -d data -o LINUX2
```

The MAC values for the unallocated inodes are saved next, using ils (TCT) and ils2mac (TCT). As previously mentioned, the ils program will output all unallocated inodes and the ils2mac program will convert the output to the same format that grave-robber produces. These commands are based on Dave Dittrich's Challenge analysis [4].

```
# for i in 1 5 6 7 8
> do
> ils images/honeypot.hda$i.dd | ils2mac > data/hda$i.ils
> done
# cat data/hda?.ils > data/body.ils
```

As the compromised system was running Linux, fls can be used to collect the MAC times of recently deleted files. As previously described, the '-m' flag with fls will output the deleted file names and attributes in the time machine format, which mactime uses.

```
# fls -m "mnt/boot/" images/honeypot.hda1.dd > data/hda1.fls
# fls -m "mnt/usr/" images/honeypot.hda5.dd > data/hda5.fls
# fls -m "mnt/home/" images/honeypot.hda6.dd > data/hda6.fls
# fls -m "mnt/var/" images/honeypot.hda7.dd > data/hda7.fls
# fls -m "mnt/" images/honeypot.hda8.dd > data/hda8.fls
# cat data/hda?.fls > data/body.fls
```

The body, body.ils, and body.fls files are concatenated together and processed. The mactime program will extract all entries with dates starting at November 7, 2000 (the day of the attack) and will translate the UID and GID to names using the original password and group files.

```
# cat data/body data/body.ils data/body.fls > data/body.all
# mactime -p mnt/etc/passwd -g mnt/etc/group -b data/body.all \
> 11/07/2000 > data/mactime.txt
```

5.1.2 Result

The generated timeline, in data/mactime.txt, has MAC times for active files (grave-robber):

```
Nov 08 00 09:51:53  350996 .a. -rwxr-xr-x 1010  users
   mnt/usr/man/.Ci/syslogd
```

unallocated inodes (ils):

```
Nov 08 00 09:51:53    1153 .a. -rwxr-xr-x 1010  users
   <honeypot.hda5.dd-dead-109801>
```

and unallocated files (fls):

```
Nov 08 00 09:51:53    1153 .a. -rwxr-xr-x 1010  users
   mnt/usr/man/.Ci/install-sshd1 <109801> (deleted)
```

As grave-robber produces entries for allocated inodes and ils produces entries for unallocated inodes, the timeline would be complete before the fls output was added. The benefit of using fls is that a filename can be associated with the output of ils. For example, in the above examples the second and third are for the same unallocated inode (109801).

Most likely, there will be more entries from ils than from fls. For example, at 09:52:53 on hda5 there are over 2400 entries for unallocated inodes. There are also 17 fls entries at the same time that can be correlated with 17 of the 2400 unallocated inodes by their inode number. The fls entries are for files in /usr/man/.Ci , /usr/man/man1, and /usr/man/man8, so this gives an investigator insight that some of the thousands of unallocated inodes were from somewhere in /usr/man/.

An alternative method of determining the file name of an ils generated entry is to use find_file (TCTUTILs).

ils output can also be used to validate output from fls. If a file is deleted, and the associated inode is reallocated to a new file, then fls will display the deleted file name with the properties of the new file. This scenario can be detected in the timeline. Every fls entry must have an accompanying ils entry for the MAC times to be considered valid. If an ils entry does not exist, then examining the file size, UID, GID, or permissions can identify the file that reallocated the inode.

5.1.3 Analysis

Many techniques can be used to analyze the timeline. The technique that will be used here is to make several passes. Each time the timeline will be broken into smaller pieces, until most events can be accounted for.

At first glance, one notices that starting at 05:02:00 (EST) thousands of files were accessed across all partitions in the course of six seconds and then it was quiet for over four hours. A first guess is that it was from an administrative cron job. At 09:25:53 (EST) some activity in /etc/ occurred and many files were changed in /usr/man/.Ci. There are also many access time entries for library files, most likely from compiling programs. At 09:52:53 there are thousands of inodes that are unallocated and according to fls some of them belong to files in /usr/man/.Ci. More files are accessed and changed until 10:03:15. An interesting entry from fls is a directory named /dev/tpack that was deleted at 09:59:14. Due to the abnormal activity in /usr/man/.Ci and /dev/tpack/, this appears to be the timeframe that the attacker was active. When one investigates the 09:25:53 to 10:03:15 time, it is observed that the am-utils, make, lpd, ypserv, ssh, ftp, and nfs-utils packages were installed, based on file names.

The timeline is a good place to identify files and inodes to examine later using Autopsy. Based on this quick survey, it is important to verify a cron job at 05:02:00, investigate the contents of the /usr/man/.Ci directory, and /dev/tpack. It is also interesting to scan the timeline for unallocated inodes with a large size and no corresponding fls entry. An obvious first choice is inode 8133 on hda8. It was modified shortly after the ftp executable was accessed, before the /usr/man/.Ci directory was changed, and is over 2MB in size.

5.2 File Browsing

Using files identified from the timeline, namely the /usr/man/.Ci directory, an example of file browsing in Autopsy will be given next. From the Main Menu, select the "File" option, the two boxes next to the "honeypot.hda5.dd" image, and press the "Enter" button. The left-hand frame contains all directories under /usr/ and the upper right contains the directories and files that are directly under /usr/. The man directory is selected in the upper right-hand window, followed by the .Ci directory. This directory was identified in the challenge as the installation directory for a rootkit.

One can easily traverse the files and subdirectories of the installation directory. Some interesting deleted files to observe are named.tgz, named.tar, ssh-1.2.27.tar, nfs-utils-0.1.9.1-1.i386.rpm, and wuftpd.rpm. All of these installation files can be viewed by selecting the name and can be recovered using icat (TCT). One also observes that script files named install-wu, install-statd, install-sshd1 were deleted and appear to install the software packages that were noticed in the timeline.

The timeline showed that at 09:52:10 (EST), system executables were copied to /usr/man/.Ci/backup, followed by an access of the /usr/man/.Ci/fix program. The fix program is part of the Linux Rootkit 4 [7] (lrk4) and modifies the CRC value and MAC times of a trojan binary to match that of the original to avoid detection. Therefore, one should examine the original location of the files in the backup directory. This can be done in Autopsy.

As an example, /bin/ps will be examined. The honeypot.hda8.dd image is loaded into Autopsy and the /bin/ directory is opened. When the ps file is selected, the contents are displayed in ASCII. As this is a binary, this is not very helpful. By selecting the "strings" tag on the lower frame, a list of the ASCII strings in the binary is shown. By scrolling through them, one notices the line /dev/ptyp. The Linux Rootkit uses a /dev/ptyp file as a configuration file. A string that shows it is based on the same program as the lrk4 trojan, *procps-1.01*, is also identified. This shows that the /bin/ps binary was replaced by a trojaned version that will hide processes based on the contents of /dev/ptyp.

It is important to document these contents, so one can select the "report" tag next to "strings" in the lower window. A new browser will open with the strings output of the ps file, along with MD5 checksums of the original binary and the strings output. The report also includes all 39 blocks that this file uses, dates, UID, GID, and permissions in an ASCII format.

5.3 Inode Browsing

Inode 8133 on hda8, identified by the timeline, will now be examined. From the Main Menu of Autopsy, select the "Inode" box and the two boxes next to "honeypot.hda8.dd". The right-hand side of the browser has a text box to enter the inode number and the left-hand side contains a menu to change browsing modes. Enter "8133" into the box on the right and press the "Display" button.

The right-hand side frame replaces itself with details on the inode. It shows that the inode is not currently allocated and that it's size is over 2MB. The blocks it has allocated are also shown, starting at 33064 until 35153. When the first block is selected, 33064, the view changes to block browsing mode and the frame breaks into an upper and lower half. The upper frame shows that this block is allocated by inode 8133 and the lower frame shows the block contents in ASCII. The block contains unprintable ASCII, so one goes back by pressing the "Back" button on the HTML browser. The contents of the next block start with the string “/src/Makefile" and contains text in the format of a makefile. The next block, 33066, is also in a makefile format and contains the comment: "Making eggdrop..." (*eggdrop* is an Internet Relay Chat (IRC) robot). Based on the format of the blocks, this inode is a tar file that installs *eggdrop*. It can be recovered and analyzed by the following command:

```
# icat images/honeypot.hda8.dd 8133 > hda8_i8133_eggdrop.tar
```

5.4 Block Browsing & Lazarus

5.4.1 Creation

Block browsing was already used with inode browsing, but it will now be used in conjunction with lazarus (TCT). To make the lazarus files, the unallocated image blocks are collected using unrm (TCT). The following commands are executed from /usr/local/forensics/chal/:

```
# mkdir laz_hda8
# unrm images/honeypot.hda8.dd > laz_hda8/hda8.unrm
# cd laz_hda8
# mkdir data
# mkdir html
# lazarus -h -w /usr/local/forensics/chal/laz_hda8/html \
> -D  /usr/local/forensics/chal/laz_hda8/data/ -H . hda8.unrm
```

lazarus does not add the current working directory when relative paths are given, so it must be fully specified or else the HTML pages will not work. The above command creates an hda8.unrm.frame.html file in the laz_hda8 subdirectory. This file should be opened in an HTML browser.

As previously mentioned, the block numbers that lazarus uses will be different then the actual block number, because it uses unrm. In this example, the lazarus (unrm) blocks will be written as Xu, for any integer X, and the Autopsy (dd) blocks will be written as Xd.

5.4.2 Analysis

lazarus lets one examine the unallocated block contents by type. Lets look for text files. The second set of text blocks, block 7899u, is examined and a makefile is identified. While lazarus tries to group blocks together of similar content, it does not always find all of the blocks. The best way is to determine if there is an inode associated with it. Autopsy will be used for this.

Block Browsing mode in Autopsy is entered from the Main Menu. The block number “7899” is entered into the text box and the unrm block type is selected from the menu. Autopsy translates the unrm block number to the dd block number and displays the contents of block 8490d. It shows that this block belongs to inode 2050, which is not currently allocated to a file. When the inode is selected, one sees that a total of four blocks were allocated by this inode (lazarus only showed three). This makefile is for the installation of *eggdrop*. It must have been created when the tar file (which was found in inode 8133) was opened. Using the inode browsing mode, one can also determine that it was deleted at 09:58:57 (EST). From within the inode browsing mode, “view as file” can be selected to display the contents of all four blocks.

5.5 Searching

The rootkit tar file has not been found yet, so a block-level search will be done to find it. The rootkit directory, /usr/man/.Ci, contained many files beginning with the name "install-", so that will be searched for. As the *eggdrop* tar file was found on hda8, that image will be searched first.

From the Main Menu of Autopsy, select "Search" and the two boxes next to "honeypot.hda8.dd". The browser has a box to enter the search string, and "install-" is entered (without the quotes). After a little while, the left-hand side frame contains 14 instances of "install-" in 11 blocks. When one selects a block, the right-hand size goes to block browsing mode and displays an upper and lower frame. If the block is allocated by an inode, it will be identified in the upper half. The lower half contains the block contents.

The first block, 96112, is part of a tar file and contains the string "install-sshd1". It is the script that was previously seen while file browsing in /usr/man/.Ci. According to the top frame, this block is not allocated to an inode, so it will not be possible to recover the entire tar file using icat. Block 96118 has three instances of the string and its contents are examined next. The first line of this block is the end of a sentence, so it appears to be part of a file that started in block 96117. The instances in 96118 are for "install-sshd" and "install-sshd1". When the "previous" button in the upper frame is used to examine block 96117, it can be seen that this block starts with the string ".Ci/install". It is a script file and the first few lines are commands that make symbolic links from bash_history files to /dev/null and copies files to a directory named backup. These commands correspond to the events that were observed in the timeline starting at 09:52:09 (EST). From the script contents and the contents of the other scripts that it calls, one can identify this script as the main installation script for the rootkit. Using the "next" and "previous" buttons one can observe the neighboring blocks to find additional interesting rootkit files.

6 Conclusion

TCTUTILs and the Autopsy Forensic Browser provide many functions that did not previously exist for FFS and EXT2FS image analysis. TCTUTILs provides support for file and directory names, while Autopsy provides an interface to make the low-level forensics tools in TCT and TCTUTILs easier to use.

Future versions of this software will attempt to integrate more low-level tools to automate basic procedures, yet allow an investigator the flexibility to perform tasks that are unique to the circumstances. This creates a flexible and open environment where an investigator can understand and confirm every step that occurs.

7 References

[1] B. Carrier, Autopsy Forensic Browser v1.01, available at: http://www.cerias.purdue.edu/homes/carrier/forensics.html

[2] B. Carrier, TCTUTILs v1.01, available at: http://www.cerias.purdue.edu/homes/carrier/forensics.html

[3] D. Dittrich, The Honeynet Forensic Challenge, available at: http://project.honeynet.org/challenge/

[4] D. Dittrich, Honeynet Forensic Challenge Analysis, available at: http://project.honeynet.org/challenge/results/dittrich/evidence.txt

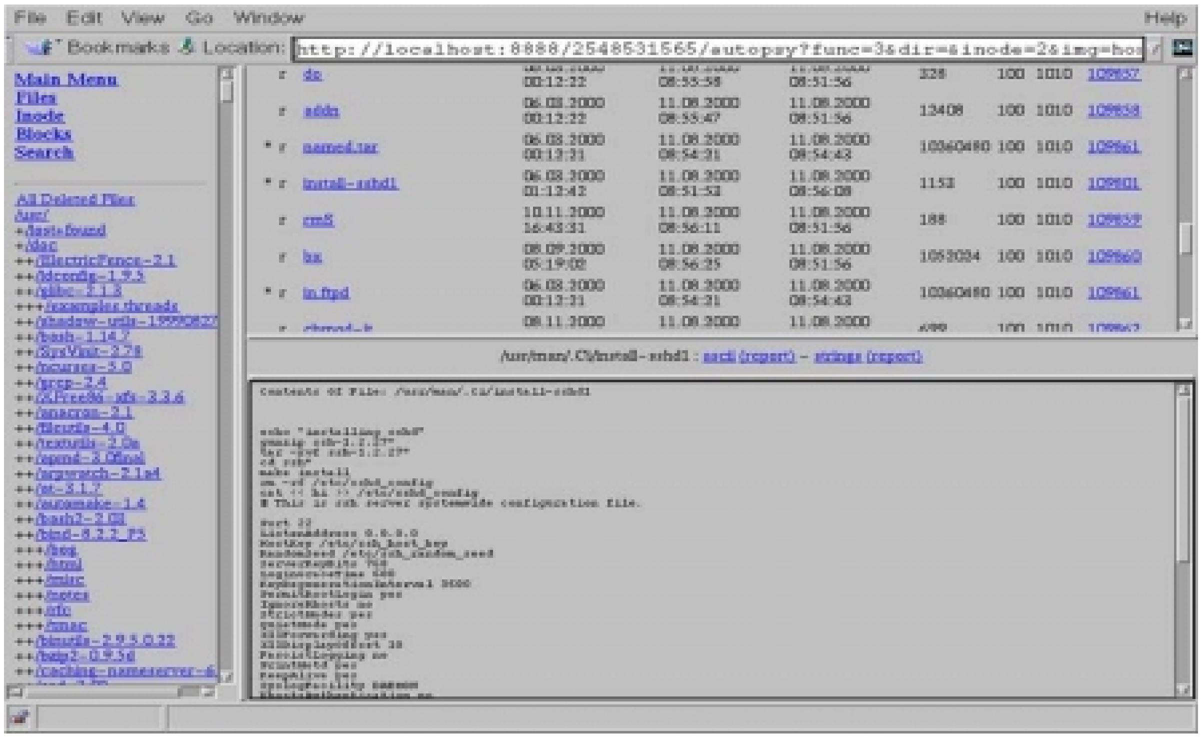
[5] D. Farmer and W. Venema, The Coroners Toolkit (TCT) v1.06, available at: http://www.porcupine.com/tct

[6] J. Holley, September 2000 Market Survey: Computer Forensics, SC Magazine, available at: http://www.scmagazine.com/scmagazine/2000_09/survey/survey.html

[7] L. Somer, Linux Rootkit 4, Nov. 26, 1998, available at: http://packetstorm.securify.com/UNIX/penetration/rootkits/lrk4.src.tar.gz

Appendix A

File and Directory Browsing Screen Shot:


File Edit View Go Window Help
Bookmarks Location: http://localhost:8888/2548531565/autopsy?func=3&dir=&inode=2&img=ho
Main Menu
Files
Inode
Blocks
Search

```
All Deleted Files
/usr/
+/lost+found
+/doc
++/ElectricFence-2.1
++/ldconfig-1.9.5
++/glibc-2.1.3
+++/examples.threads
++/shadow-utils-19990827
++/bash-1.14.7
++/SysVinit-2.78
++/ncurses-5.0
++/grep-2.4
++/XFree86-xfs-3.3.6
++/anacron-2.1
++/fileutils-4.0
++/textutils-2.0e
++/apmd-3.0final
++/argwatch-2.1a4
++/at-3.1.7
++/automake-1.4
++/bash2-2.03
++/bind-8.2.2_P5
+++/bog
+++/html
+++/misc
+++/notes
+++/rfc
+++/tmac
++/binutils-2.9.5.0.22
++/bzip2-0.9.5d
++/caching-nameserver-6
```

```
    r  dp              00:12:22      08:55:58      08:51:56    328        100  1010  109837
    r  addn            06.08.2000    11.08.2000    11.08.2000
                       00:12:22      08:55:47      08:51:56    12408      100  1010  109838
  * r  named.tar       06.08.2000    11.08.2000    11.08.2000
                       00:13:21      08:54:21      08:54:43    10360480   100  1010  109861
  * r  install-sshd1   06.08.2000    11.08.2000    11.08.2000
                       01:13:42      08:51:52      08:56:08    1153       100  1010  109801
    r  rm8             10.11.2000    11.08.2000    11.08.2000
                       16:43:31      08:56:11      08:51:56    188        100  1010  109839
    r  bx              08.09.2000    11.08.2000    11.08.2000
                       05:19:02      08:56:25      08:51:56    1053034    100  1010  109860
  * r  in.ftpd         06.08.2000    11.08.2000    11.08.2000
                       00:13:21      08:54:21      08:54:43    10360480   100  1010  109861
    r  chmod-it        08.11.2000    11.08.2000    11.08.2000    490        100  1010  109862
```

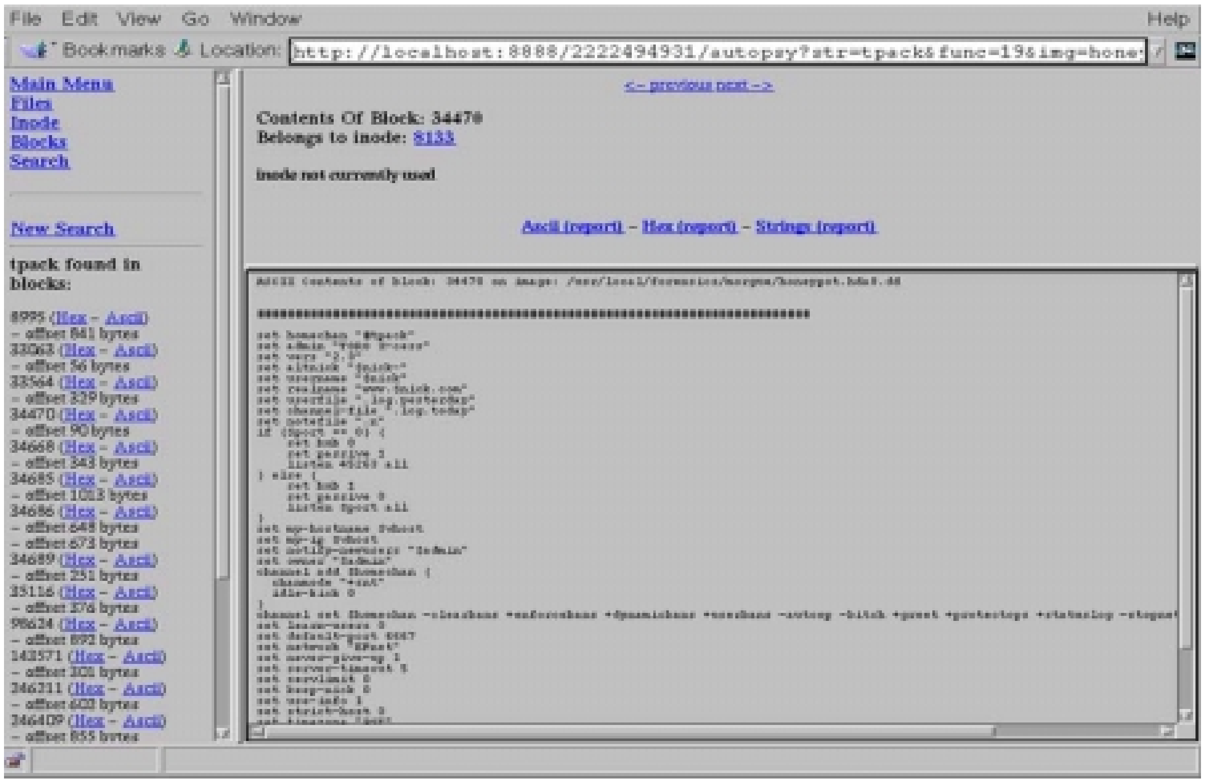
/usr/man/.Ci/install-sshd1 : ascii (report) - strings (report)

```
Contents Of File: /usr/man/.Ci/install-sshd1


echo "installing sshd"
gunzip ssh-1.2.27*
tar -xvf ssh-1.2.27*
cd ssh*
make install
rm -rf /etc/sshd_config
cat << hi >> /etc/sshd_config
# This is ssh server systemwide configuration file.

Port 22
ListenAddress 0.0.0.0
HostKey /etc/ssh_host_key
RandomSeed /etc/ssh_random_seed
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes
IgnoreRhosts no
StrictModes yes
QuietMode yes
X11Forwarding yes
X11DisplayOffset 10
FascistLogging no
PrintMotd yes
KeepAlive yes
SyslogFacility DAEMON
RhostsRSAAuthentication no
```

Block Search Screen Shot:


File Edit View Go Window Help
Bookmarks Location: http://localhost:8888/2222494931/autopsy?str=tpack&func=19&img=hone
Main Menu
Files
Inode
Blocks
Search
New Search
<- previous next ->

tpack found in blocks:

```
8995 (Hex - Ascii)
- offset 841 bytes
33063 (Hex - Ascii)
- offset 56 bytes
33564 (Hex - Ascii)
- offset 329 bytes
34470 (Hex - Ascii)
- offset 90 bytes
34668 (Hex - Ascii)
- offset 343 bytes
34685 (Hex - Ascii)
- offset 1013 bytes
34686 (Hex - Ascii)
- offset 648 bytes
- offset 673 bytes
34689 (Hex - Ascii)
- offset 251 bytes
35116 (Hex - Ascii)
- offset 376 bytes
98624 (Hex - Ascii)
- offset 892 bytes
143571 (Hex - Ascii)
- offset 301 bytes
246311 (Hex - Ascii)
- offset 603 bytes
246409 (Hex - Ascii)
- offset 855 bytes
```

Contents Of Block: 34470
Belongs to inode: 8133

inode not currently used

Ascii (report) - Hex (report) - Strings (report)

```
ASCII Contents of block: 34470 on image: /usr/local/forensics/morgue/honeypot.hda8.dd


##########################################################################

set homechan "#tpack"
set admin "[illegible]"
set vers "2.3"
set altnick "tnick-"
set username "tnick"
set realname "www.tnick.com"
set userfile ".log.yesterday"
set channel-file ".log.today"
set notefile ".n"
if {$port == 0} {
     set bnb 0
     set passive 1
     listen 45263 all
} else {
     set bnb 1
     set passive 0
     listen $port all
}
set my-hostname $vhost
set my-ip $vhost
set notify-newusers "$admin"
set owner "$admin"
channel add $homechan {
  chanmode "+snt"
  idle-kick 0
}
channel set $homechan -clearbans +enforcebans +dynamicbans +userbans -autoop -bitch +greet +protectops +statuslog -stopnet
set learn-users 0
set default-port 6667
set network "EFnet"
set never-give-up 1
set server-timeout 5
set servlimit 0
set keep-nick 0
set use-info 1
set strict-host 0
set timezone "GMT"
```