SANS Digital Forensics and Incident Response Blog | FreeBSD Computer Forensic Tips & Tricks

Hal Pomeranz, Deer Run Associates

While Linux seems to have captured much of the mind-share for Unix-like operating systems, the fact is that there are an awful lot of BSD machines out there, particularly in web-hosting and other Internet-facing environments. So you're likely to run into one of these systems during an incident response or digital forensics investigation at some point. If you've only ever analyzed Linux systems, you may encounter a few bumps in the road when you start looking at your first BSD system. In an effort to smooth out some of those potholes, I'm going to demo a few useful techniques using a sample FreeBSD image I created.

BSD Disk Labels

Let's suppose somebody just handed you a raw disk image that they took from a FreeBSD machine. Not being Unix savvy, all they can do is hand you the raw disk image and the associated checksum value. The rest is up to you.

After verifying the image integrity, you decide you're going to start by dumping out the DOS partition table:

```
# mmls -t dos freebsd.img
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

 Slot    Start        End          Length       Description
00:  -----   0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000001   0000000062   0000000062   Unallocated
02:  00:00   0000000063   0033554114   0033554052   FreeBSD (0xA5)
03:  -----   0033554115   0033554431   0000000317   Unallocated
```

The DOS partition table shows us that there's only a single partition on this disk (plus a couple of unallocated areas which are going to turn out to be just wasted space). This already might make you a little suspicious- most Unix-like operating systems reserve a raw disk partition for a swap area.

In fact, the DOS partition table isn't telling us the whole story. BSD systems set up another partition table- generally referred to as the *BSD disk label*— at the start of the main BSD disk partition. mmls will happily parse the BSD disk label if we specify "-t bsd" and as long as we give it the correct sector offset value:

```
# mmls -t bsd -o 63 freebsd.img
BSD Disk Label
Offset Sector: 63
Units are in 512-byte sectors

 Slot    Start        End          Length       Description
00:  -----   0000000000   0000000062   0000000063   Unallocated
01:  00      0000000063   0001048638   0001048576   4.2BSD (0x07)
02:  02      0000000063   0033554114   0033554052   Unused (0x00)
03:  01      0001048639   0002029262   0000980624   Swap (0x01)
```

```
04:  03      0002029263   0004615886   0002586624   4.2BSD (0x07)
05:  04      0004615887   0005664462   0001048576   4.2BSD (0x07)
06:  05      0005664463   0033554114   0027889652   4.2BSD (0x07)
07:  -----   0033554115   0033554431   0000000317   Unallocated
```

Now that looks like a more standard partition layout for a Unix-like system. We can see multiple partitions and a swap area.

Mounting UFS Partitions

Let's try looking at that first partition with fsstat to see what kind of file system we're dealing with:

```
# fsstat -o 63 freebsd.img
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: UFS 2
Last Written: Mon Jan 25 03:58:20 2010
Last Mount Point: /
Volume Name:
System UID: 0
[...]
```

UFS2 is the most common file system for modern FreeBSD releases. The good news is that Linux has excellent support for UFS file systems, so you can do analysis of FreeBSD images on a Linux machine.

However, there are a couple of special options that you need to use when mounting a UFS file system on a Linux box. You need to not only specify the file system type with "mount -t ...", you also have to supply the "-o ufstype=..." option in order to specify the exact type of UFS file system you're dealing with. And of course you'll have to calculate the correct "offset=" value for the mount command (it's in bytes, not sectors, as I covered in my previous article on the subject).

Putting it all together, calculating the offset and doing the mount command might look something like this:

```
# expr 63 \* 512
32256
# mount -t ufs -o ufstype=ufs2,ro,loop,offset=32256 freebsd.img /mnt
# ls /mnt
bin   cdrom   COPYRIGHT  dist     etc  libexec  mnt   rescue  sbin  tmp  var
boot  compat  dev        entropy  lib  media    proc  root    sys   usr
```

First we use expr to calculate the byte offset of the start of the first partition: it's the sector offset (63, as shown in the mmls output) times the sector size (512 bytes/sector, also shown in the mmls output).The resulting value, 32256, is going to get plugged into the mount command with "-o ...,offset=32256".

Next we set up our mount command. We specify the UFS file system and specific UFS type with "-t ufs -o ufstype=ufs2,...". Obviously, the read-only option ("ro") is important from a computer forensics perspective. The "loop" argument means we'll be using a loopback mount to mount a disk image file as a file system. Then we have our offset argument, the name of the disk image file, and where we want this partition mounted. We can then get a directory listing of our mount point and confirm that the mount succeeded and we can see files.

Mounting the other disk partitions is just a matter of repeating our original mount command with different offsets and mount points. However, we'd like to know where those partitions should be mounted in order to form a file system that correctly replicates the original layout of the system. You could get the last mount point information by running fsstat on each slice, but since we have the root file system already mounted, we can just dump the /etc/fstab file from the disk image:

```
# cat /mnt/etc/fstab
# Device        Mountpoint    FStype    Options        Dump    Pass#
/dev/ad0s1b        none        swap   sw        0    0
/dev/ad0s1a        /           ufs    rw        1    1
/dev/ad0s1e        /tmp        ufs    rw        2    2
/dev/ad0s1f        /usr        ufs    rw        2    2
/dev/ad0s1d        /var        ufs    rw        2    2
/dev/acd0          /cdrom      cd9660    ro,noauto    0    0
```

BSD uses letters rather than numbers to distinguish disk slices, so you'll have to do a little mental conversion. Going by the above fstab, BSD partition #4- /dev/ad0s1d in the fstab- is /var, partition #5- /dev/ad0s1e- is /tmp, and partition #6- /dev/ad0s1f- is /usr. So now you know where to put everything when you mount it.

About Blocks and Fragments

Let me show you one more interesting aspect of computer forensics on a BSD system. Here's an excerpt from the fsstat output on the /var partition from our image:

```
# fsstat -o 2029263 freebsd.img
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: UFS 2
Last Written: Mon Jan 25 04:01:14 2010
Last Mount Point: /var
[...]
CONTENT INFORMATION
--------------------------------------------
Fragment Range: 0 - 646655
Block Size: 16384
Fragment Size: 2048
Num of Avail Full Blocks: 77661
Num of Avail Fragments: 315
[...]
```

Notice that the block size for the file system is 16K, but the fragment size is 2K. In traditional Berkeley file systems, each block is made up of smaller fragments which are fully addressable and can be used to store individual files that are smaller than the block size. If you look at a typical EXT file system on Linux, you'll see that the block size and the fragment size are the same, making the smallest disk unit on EXT effectively be an entire block. Of course, EXT blocks are generally smaller (usually 4K), so this isn't quite as wasteful as you might think.

Why is this difference between the block size and fragment size relevant to your computer forensic investigations on FreeBSD? Well, if you've got a string of interest at a particular byte offset, is your tool of choice going to address the location of that string in terms of blocks (divide by 16K) or fragments (divide by 2K)?

Turns out that the Sleuthkit at least addresses things in terms of fragments. Let me prove it to you. Here's our string hit:

```
6148096 # This file lists authorizations for user haldaemon
```

We're going to divide the byte offset by the fragment size and then use blkcat to dump the fragment.

If I'm telling you the truth, we should see our string of interest in the blkcat output:

```
# expr 6148096 / 2048
3002
# blkcat -o 2029263 freebsd.img 3002
# This file lists authorizations for user haldaemon
#
# File format may change at any time; do not rely on it. To manage
# authorizations use polkit-auth(1) instead.

scope=grant:action-id=org.freedesktop.policykit.read:when=1263385148:granted-by=0
```

Bingo! From here we could work back to to the actual file name using the TSK tools, but that's an article for another day perhaps.

Conclusion

I hope this article helps illuminate some of the differences between FreeBSD and Linux analysis. There are other differences, of course, largely dealing with different file system layouts ("Where were those crontabs again? Oh yeah, /var/cron/tabs/..."). But this should be enough to get you moving in the right direction.

Hal Pomeranz is an Independent IT/Security Consultant, a SANS Institute Faculty Fellow, and a GCFA. And, yes, he swings both ways... when it comes to Unix-like operating systems, that is. Hal will be teaching Security 508: Computer Forensics, Investigation, and Response at SANS Virginia Beach, May 24-29.