

## Incident Response Tools For Unix, Part Two: File-System Tools

*Holt Sorenson* 2003-10-17

This is the second article in a three part series on tools that are useful during incident response and investigation after a compromise has occurred on a Linux, OpenBSD, or Solaris system. The [first article](#) focused on system tools, this one focuses on file system tools, and the next article will discuss network and other tools. The information used in these articles is based on [OpenBSD 3.2](#), [Debian GNU/Linux 3.0 \(woody\)](#), [RedHat 8.0 \(psyche\)](#), and [Solaris 9](#) (aka Solaris 2.9 or SunOS 5.9). The tools focused on are generally tools that are available with the operating system, although there are some that may not be native to a given system that are discussed as well. If a tool that is discussed isn't available on the operating system you're using, the information on acquiring tools in the references section [\[1\]](#) might help you out.

The tools that are covered in this article are all tools that execute in user-space. If an attacker has compromised the system and installed a kernel module that hides his activities, or a root-kit that changes the binaries on the system, the results that the tools below provide are likely to not be accurate. This is one of the reasons that offline analysis should be performed after data from the live system has been secured. This live data shouldn't be trusted as valid until it has been corroborated. Part of the reason that so many tools are being covered is to familiarize the reader with multiple tools that do similar things so that one can check a tool's version of reality with other tools. It's important that those responding to incidents not prematurely rule out possibilities and that they remain skeptical.

It can't be emphasized enough that these tools should be executed from read-only media [\[2\]](#) or on a secure system that is reserved for offline analysis. Using read-only media ensures that the tools haven't changed since they were stored on the read-only media, making it less likely that an attacker has compromised the tools.

When responding to an incident, security personnel need to already have decided what their priorities are for capturing data. The data on the system has a propensity for change. Incident response teams need to debate whether it is more important to immediately take a machine offline and image the file system, or to capture live data such as processes that are currently executing, established network connections, memory allocated by processes, and users that are logged in. The longer a system is kept on the network after a compromise, the more likely it is that there will be more damage caused by the attacker to remote or local systems. Note that these statements aren't asserting that one should capture live data and not create file system images--if you decide the live data is important, you still need to image the file system after capturing the live data.

Law enforcement should be consulted to verify that the procedures that the security team has developed will facilitate any necessary law enforcement investigation. Organizations such as the [HTCIA](#) (US) and the [NHTCU](#) (UK) can help technical personnel get into contact with law enforcement officials that are responsible for computer related criminal investigations in their area.

The [previous](#) article in the series finished at the point where a password cracker, John the Ripper, had been found on a compromised web server after a webmaster had reported performance issues. In this article we explore how file and file system tools can be used to capture information that may shed light on how the host got compromised.

### Sum packages need to be checked

The attacker began to use the web server to crack passwords at 03:17. 03:17 is a time from which to start the search, but this time shouldn't necessarily be viewed as the time that the attacker compromised the system. Attackers frequently use automated programs that scan and attack vulnerable systems. It is possible that the attacker is just getting around to visiting a system that an automated tool hacked some time ago.

Lets dig into some tools that can be used to help detect changes on the file system by an attack. Some of the tools that will be covered are included with a stock operating system. Other third-party tools are installed and customized to the operating system.

On Solaris and Linux, there are some native commands that help compare the current operating system install against the metadata contained in the package database. The commands are: **rpm -Vva** (RedHat 8.0), **pkgchk -vn** (Solaris 9), and **debsums -ac** (Deb 3.0). Both the rpm and the Debian format[3] can use RFC2440 (OpenPGP) compliant signatures to authenticate the packages. Both also use MD5 hashes as their "checksums". On Solaris, **pkgchk** uses the System V (SYSV) sum algorithm to verify that binaries haven't been modified. The SYSV sum algorithm will generally detect changes to file contents that don't change the file size. However, it is unacceptable to depend on the SYSV sum algorithm for integrity. It's only slightly more robust than relying on file size and timestamps. Any of you who have spent some time in a hex editor or have used the **touch** command know how easy it is to change the file contents without changing the size or how easy it is to change the timestamps on the file's inode. On Debian GNU/Linux distributions, you can specify a set of packages to check against. If you mounted a read-only set of packages on /mnt, the command that you use to verify against the package files is: **debsums -cagp /mnt/\***. The equivalent **rpm** command is: **rpm -Vvp /mnt/RedHat/RPMS/\***. **rpm** can also verify against a different rpm database than the one on your system. Use **rpm -Vva --dbpath <some\_path>** to verify against the alternate database. The equivalent command on a Debian system is: **debsums -cagd <some\_path>**. These databases should reside on read-only media and should come from a trusted system. If you want a ridiculous amount of verbosity from **rpm**, add another **-v**. The output of these commands can be used to find changes that might have been caused by an attacker.

#### package tools verifying current file system state against package metadata:

```
{redhat80} $ rpm -vVa
[ snip lots of output ]
..5....T  /bin/ls
[ snip lots of output ]

{solaris9} $ pkgchk -vn
[ snip lots of output ]
ERROR: /usr/bin/ls
      modtime <04/06/02 10:54:41 PM> expected <09/08/03 01:04:01 AM> actual
      file cksum <63074> expected <63042> actual
/usr/bin/ls
[ snip lots of output ]

{debian30} $ debsums -ac
[ snip lots of output ]
bin/ls
[ snip lots of output ]
```

Comparing the package database to the currently installed system generally only gets you so far, however. On systems that have been heavily customized, the metadata in the packaging system and the actual state of the file system can provide two disparate views of reality. Using package tools to compare current file system state against the package database might not net you the results you desire. OpenBSD has a tool called **mtree** [4] that can use hash functions [5] to give assurance that files on the file system haven't changed since its database was last updated. If an attacker trojans or modifies files whose hashes have been stored in **mtree**'s database, then the changes will be detected by **mtree**. Similar functionality can be achieved by installing third-party tools [6] such as **AIDE**, **integrit**, **Osiris**, and **tripwire** on Solaris and Linux. Tools [7] such as **changedfiles**, **dnotify**, and **FAM** use kernel modules or poll the file system to detect changes as (or close to, when the tool uses polling) they are made to the file system. These tools result in more rapid notification of changes than the file system integrity checking tools. This is because the file system integrity checking tools are ran periodically because the hash functions they use are computationally intensive. While it should be obvious, these tools would need to be installed and configured prior to a compromise.

When these automated tools aren't available or when you need to dig deeper, there are some other tools that can be used during a response to an incident. These tools are usually used on files, although on Solaris, they can run on directories as well. The values that these tools display aren't useful unless you have a trusted system on which you can run the same tool and compare the results. If you trust [The Shmoo Group](#), you can consult the [Known Goods](#) database to find hashes and file sizes of binaries. The tool that provides the MD5 hash function on Linux systems is called **md5sum** (this is part of the the [FSF's coreutils](#) software package). On OpenBSD, the analogous tool is **md5**. OpenBSD also provides **sha1** and **rmd160**, based on the SHA1 and RIPEMD160 hash functions, respectively. The [OpenSSL](#) library provides hash functions that can be run from the command line application **openssl**. **openssl dgst <some\_file>** uses the MD5 hash function by default. Use **openssl dgst -?** to see the usage for the **openssl dgst** command and the different hash algorithms it supports. One can also run the commands without including the token 'dgst'. For example, if you required the MD5 hash of a file, you could run **openssl md5 <filename>**. Another utility, **shash** [8], provides "checksums" using the hash function algorithms previously mentioned (see **shash -l** for the list). It can also compute checksums using a couple of other algorithms, as well.

Commands that provide checksums that aren't based on hash functions shouldn't be relied on for incident response. Two examples of such commands are **cksum** and **sum**. Checksum algorithms (CRC\*, BSD, and SYSV sum) can't be trusted to provide the level of assurance that hash functions are accorded. In fact, checksum algorithms shouldn't be trusted for forensics purposes at all. Checksum algorithms are designed to detect errors in data being transmitted or stored, but are not designed to protect data from being altered by a persistent and/or malicious attacker. They don't have the property of being one-way, and it is easy to find multiple sets of data that have the same checksum. Even if you took a checksum of some data and then stored it with the data on read-only media to ensure that the data being stored wasn't tampered with, the defense could trivially create other sets of data that had the same checksum. The defense could then argue that the data you stored wasn't the data that was relevant to the case being heard because the checksum couldn't provide assurance that the data was the data relevant to the case. The checksum algorithms that sum and cksum depend on map an infinity of possibilities to a set that is too small. Tools that rely on MD5 map the infinity of possibilities to  $2^{128}$  possibilities. Tools that rely on SHA1 map the infinity of possibilities to  $2^{160}$ . Both MD5 and SHA1 map in such a way that, in practical terms, it is difficult to find two sets of data for which the result that a hash function produces are the same. The checksum algorithms are designed to protect against accidental corruption

of data, not to protect against malicious attacks. This is why using a tool that depends on hash functions such as MD5 or SHA1 is a necessity.

#### Hash/Checksum tools in action:

```
{deb30} $ md5sum /bin/ls
a5c720b6776331b9695d9a1f4f5c2194 /bin/ls
{deb30} $ openssl dgst /bin/ls
MD5(/bin/ls)= a5c720b6776331b9695d9a1f4f5c2194
{deb30} $ shash /bin/ls
# MD5 HASH
a5c720b6776331b9695d9a1f4f5c2194 /bin/ls
{deb30} $ cksum /bin/ls
2890986056 43784 /bin/ls
{deb30} $ sum -r /bin/ls
56701 43
{deb30} $ sum -s /bin/ls
6968 86 /bin/ls
```

## Hey MAC, what time ya got?

Unix (and unix-like) file systems store a set of timestamps in their file system metadata. These timestamps are called the modify, access, and change (MAC) times. These timestamps are all stored in a file system structure called an inode. In addition to these timestamps, the inode contains information about the file such as its file type, permissions, owner, group, size, number of hard links to the file, and the data blocks that the contents of the file occupies. Inodes also exist for directories. The directory inodes contain information such as the name of files inside the directory and the inode numbers that contain the file information.

The mtime reflects the time at which the file's data was last modified. System calls such as write, truncate, and mknod change mtime. The ctime reflects the time at which the file's inode was last changed. The atime reflects the time at which the file's data was last accessed. atime is changed by the system calls such as execve, read, mknod, utime, and pipe. More details on atime, mtime, and ctime can be found in your system's stat(2) man page. This man page will also contain the system calls that change a given timestamp.

Various commands change the MAC times in different ways. The table below shows the effects that some common commands have on MAC times. These tables were created on Debian 3.0 using an ext2 file system contained in a flat file mounted on a loopback device. If you find that you are mounting lots of images over the loopback device on your Linux system, you can have your bootloader pass the parameter 'max\_loop=255' to the kernel prior to booting. Then you will be able to mount up to 255 images, instead of being limited to the default, eight. Once the file system was mounted on the loop back device, it was examined with **debugfs**. Experimenting[9] with your own system to verify the information in the tables below is encouraged. These tables can serve as a general guide, however.

#### How common commands change MACtimes for a directory (foo):

Action	atime	ctime	mtime
--------	-------	-------	-------

creation (mkdir foo)	X	X	X
directory move (mv foo bar)		X	X
file creation (touch foo/foo)		X	X
file creation (dd if=/dev/zero of=foo/foo count=1)		X	X
list directory (ls foo)	X		
change directory (cd foo)			
file test (-f foo)			
file move/rename (mv foo foo_mvd)		X	X
permissions change (chmod/chown <some_perm> foo)		X	
file copy (cp foo_mvd foo)		X	X
file edit (vim foo)		X	X
file edit (emacs foo)	X	X	X
file edit (nvi/nano foo)			

### How common commands change MACtimes for a file (f1):

Action	atime	ctime	mtime
creation (touch foo)	X	X	X
creation (dd if=/dev/zero of=foo count=1)	X	X	X
rename (mv foo bar)			
permissions change (chmod <some_perm> foo)		X	
copy (cp foo bar)	X		
copy overwrite (cp bar foo)		X	X
append (cat >> foo)		X	X
overwrite (cat > foo)		X	X
truncate (cp /dev/null foo)		X	X
list file (ls foo)			
edit (vim/emacs/xemacs/joe/jed foo)	X	X	X
edit (ed/nvi/vi (sun)/vi (obsd)/nano/pico foo)	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>
1 - all times changed, but atime is slightly older than mtime and ctime			

The **ls** command can be used to show the modify, access or change times of files. The following table shows various **ls** commands that sort in reverse order by mtime, atime, or ctime. This causes **ls** to list the most recent times last.

### displaying MACtimes using ls:

	Linux (ls from GNU fileutils)	OpenBSD	Solaris
--	-------------------------------	---------	---------

mtime	ls -latr --full-time	ls -latTr	ls -latr
atime	ls -laur --full-time	ls -lauTr	ls -laur
ctime	ls -lacr --full-time	ls -lacTr	ls -lacr

The **find** command is one that can be quite useful. **find** can be used to search the file system for files that have been changed, accessed, or modified using the `-ctime`, `-atime`, and `-mtime` arguments. Be aware that the `atime` will change on directories as you run **find**. If the purpose of using **find** is more formal than exploration, you should be working on a read-only image.

Most Linux systems include GNU **find**[\[10\]](#), from the `coreutils` package. The GNU version of **find** is quite powerful because of the variety of features it offers to users. A time range can be specified and **find** will print the timestamps on the data in that range. If you want to search for files modified more than two, but less than seven days ago and print their `mtime`, `ctime`, and `inode`, you can use:

```
# find / \( -mtime +2 -a -mtime -7 \) -a -printf "M:%t C:%c %i\t%p\n"
```

If you don't have GNU `find`, you can use the **stat** command, assuming you are using Linux. If you don't have access to the **stat** command, but do have **perl**, you can use **perl** as a `stat(2)` [wrapper](#):

```
# find / \( -mtime +2 -a -mtime -7 \) | perl -ne
    'chomp; ($i,$m,$c)=(stat)[1,9,10];printf"M:%s\t%i\t$_\n",localtime($m).
C:".localtime($c)'
```

One possible attack vector for compromising a \*nix host is to use a broken `suid` or `sgid` binary to escalate privilege. Attackers also sometimes leave `suid` root shells on the file system for their convenience. **find** can be used to hunt down such files. To search for `suid` and `sgid` files and display their information, use:

```
find / -perm -6000 -ls
```

A few more examples of using **find** appear below:

### the find command in action:

```
##==
##== find all files which have had their status changed in
##== the last 24 hrs (display ctime, inode, and filename)
# find / -ctime -1 -printf "%c %i\t%p\n"
Fri Sep 7 11:55:14 2003 174945 /var/lib/rpm
Fri Sep 7 11:55:14 2003 174946 /var/lib/rpm/__db.001
Fri Sep 7 11:55:17 2003 174947 /var/lib/rpm/__db.002
Fri Sep 7 11:55:17 2003 174948 /var/lib/rpm/__db.003
Fri Sep 7 11:55:55 2003 160125 /var/lib/random-seed
Fri Sep 7 11:55:16 2003 222706 /var/log
Fri Sep 7 12:17:05 2003 224545 /var/log/messages
[ output deleted ]
##==
##== find all files which have had their status changed from
##== 5 to 30 (inclusive) days ago and display the ctime, inode, and filename
# find / -ctime +4 -ctime -31 -printf "%c %i\t%p\n"
```

```
Sat Aug 30 19:49:52 2003 414661 /boot/System.map-2.4.20-20.8bigmem
Sat Aug 30 19:49:52 2003 414662 /boot/config-2.4.20-20.8bigmem
Sat Aug 30 19:49:52 2003 414663 /boot/module-info-2.4.20-20.8bigmem
Sat Aug 30 19:49:53 2003 414664 /boot/vmlinux-2.4.20-20.8bigmem
Sat Aug 30 19:49:53 2003 414665 /boot/vmlinuz-2.4.20-20.8bigmem
[ output deleted ]
```

## If I had a photograph of you... something to remind me...

The command most commonly used to create file system images on Linux and Unix is the **dd** command. **dd** creates a bit-by-bit binary image of its input and saves it to its output. **dd** doesn't display its progress and can be slow when its input block size is different from its output block size. A friendlier version of **dd**, **sdd**[11], doesn't suffer from these drawbacks. **sdd** can also display the current statistics when it is sent a SIGQUIT (usually bound to ^ \) and it displays a realistic view of the amount of data processed (partial blocks aren't considered full blocks).

Prior to invoking **dd** on a file system, you should run a utility that will generate a hash of the file system to be imaged. Assuming you are on Linux and are interested in creating an image of /dev/hda, you can run **md5sum /dev/hda** and then start **dd**. After **dd** completes, the same utility should be run on the saved image. If the values don't match, the examiner should look into why there is a discrepancy. On OpenBSD and Solaris, the raw devices should be used when engaging in incident response work.

On Linux, the utility **e2image** can be used to create images of the ext2 and ext3 file systems. **e2image** interprets the file system that is being imaged, instead of saving raw bits. **e2image** may not save data that clever attackers have stored outside of file system structures on the disk. **e2image** creates "raw" and "normal" images, both of which are created as sparse files to conserve disk space. This causes a file system image that is created with **e2image** to have a different hash than the file system stored on the hard disk, making it difficult to have assurance that the data you want has been captured. These reasons lead to the conclusion that **e2image** shouldn't be used in place of **dd** for forensic images.

Another utility that can be used to create system images, **partimage**[12], can save image files to a **partimage** server via SSL. It currently supports ext2, ext3, ReiserFS, JFS and XFS. It also supports FAT16/32 and HPFS (OS/2). Work is in progress on UFS (Solaris, \*BSD), HFS (MacOS), and NTFS.

**partimage** only saves used blocks of the partition being imaged. Like **e2image**, it is possible that an image that is created using **partimage** doesn't accurately represent the state of the disk at the time the image was created. Examiners should also be aware that the popular PC product, Ghost, can create images that are not suitable for forensics. One should carefully review the manual for the version of Ghost you have to make sure you can use it and then test with it, prior to using it in a critical situation.

One of the article reviewers asked why I would include tools that aren't appropriate for forensics work in this article. I chose to include such tools to make the following point:

When selecting a tool[13] for creating forensics images, it is necessary to select a tool that will provide an accurate representation of the state of the disk at the time the image is created.

When a tool claims to make an image of a file system, you need to figure out how the tool author is defining "image". An empirical method for figuring out the definition of image is to:

1. Take a test machine's file systems offline
2. run **md5sum** or a tool that implements SHA1 on the device that contains a file systems
3. Use your imaging tool to save the data
4. Run **md5sum** or a tool that implements SHA1 on the image

If the fingerprints don't match, don't use the tool for forensics purposes.

## Would you like some file system to go with that debugger?

Sometimes command line tools such as **ls**, **stat**, and **find** don't provide enough visibility into the state of the file system you're trying to analyze. Perhaps you want to look at an image of a file system that you've created with **dd**, maybe you're worried that a rootkit has been installed and that **ls** has been trojaned, or you think there are some deleted, yet recoverable, files left on a file system that have interesting data.

All three operating systems include a file system debugger. The debugger is called **fsdb** on OpenBSD and Solaris. Solaris's debugger is the most arcane, and takes some time to get used to. **fsdb** on OpenBSD and **debugfs** for ext2 and ext3 offer integrated help that can be accessed by typing 'help'. Interactive file system debuggers for other popular Linux file systems such as JFS and reiserfs are currently not available. **xfs\_db** is a file system debugger that is available for XFS. It is not covered in this article because it currently isn't a file system that is in wide use by the operating systems discussed in this article.

Linux's ability to mount foreign file systems is quite useful for read-only forensics work. \*BSD's ffs and Solaris's ufs can be mounted using the ufs kernel module. Filesystems that are available for Linux such as ext2, ext3, JFS, reiserfs, and XFS can be mounted. MSDOS (fat16), fat32, and most NTFS images are also supported. Linux doesn't include file system debuggers for foreign file systems, however.

You can find out the file systems that are supported on your currently running Linux instance using **find /lib/modules/`uname -r` /kernel/fs/\* -type f | grep -v 'nls\'**'. To see what file systems are currently loaded, use **grep -v '^nodev' /proc/filesystems**. For more information on mounting file systems, see mount(8).

To mount an image in read-only mode use: **mount -t ext2 -o ro,loop=/dev/loop0 /var/tmp/2003\_02\_17\_attack.bin /mnt**. You can now examine the newly mounted file system using tools supplied by the OS. As you examine the image, you will be changing the access times of files and directories that you manipulate. These changes will only take place in memory. It is critical that you remember to include the read-only (ro) option to -o with your mount command so that changes aren't committed to the image file. In addition, it is wise to work with copies of images that you have verified are bit-for-bit copies. A tool like **md5sum** is sufficient for making sure that the copy is the same as the original. Working with a copy ensures that the original image won't be modified. More information on working with loop devices is available in the "THE LOOP DEVICE" section of mount(8).

## mounting a file system image in Linux:

```

##==
##== mount the image read-only so that the image doesn't change on disk
# mount -o ro,loop=/dev/loop0 /var/space/images/2003_02_17_linux.bin /mnt
##==
##== list of files
# ls -la /mnt
total 146
drwxr-xr-x  19 root    root      1024 Feb 17  2003 .
drwxr-xr-x  19 root    root      4096 Sep 12 11:55 ..
-rw-r--r--   1 root    root         0 Sep 12 11:55 .autofsck
-rw-----   1 root    root      186 Sep  6 19:58 .bash_history
drwxr-xr-x   2 root    root     2048 Feb  8  2003 bin
drwxr-xr-x   3 root    root     1024 Sep  6 19:59 boot
drwxr-xr-x  20 root    root    116736 Feb 17  2003 dev
drwxr-xr-x  41 root    root     3072 Sep 12 14:13 etc
[ output deleted ]
##==
##== unmount /mnt
# umount /mnt

```

To use **debugfs**, you type **debugfs <path\_to\_image>**. Many of the commands that you are used to using while in a shell are emulated. If you get stuck, use the **help** command at the debugfs: prompt to get pointed in the right direction. If that isn't sufficiently helpful, use **man debugfs**. **debugfs** starts in read-only mode, so it is less likely to change your image as you work with it. However, you should still work with a copy and treat the original in a sacrosanct manner. An example of using **debugfs** follows:

#### using debugfs on an ext2 file system image in Linux:

```

##==
##==
# debugfs /var/space/images/2003_02_17_openbsd_attack.bin
debugfs 1.27 (8-Mar-2002)
##== show file system statistics
debugfs: stats
Filesystem volume name:
Last mounted on:
Filesystem UUID:          93bfa3ee-d684-4d07-a5d0-1654f488aabf
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      filetype sparse_super
Filesystem state:        clean
Errors behavior:         Continue
Filesystem OS type:      Linux
Inode count:              65536
Block count:              262144
Reserved block count:    13107
Free blocks:              214567
Free inodes:              57335
First block:              1
Block size:               1024
Fragment size:           1024

```

```

Blocks per group:      8192
Fragments per group:  8192
Inodes per group:     2048
Inode blocks per group: 256
Last mount time:      Fri Jan  9 14:13:42 2003
Last write time:      Fri Feb 17 23:13:04 2003
Mount count:          1
Maximum mount count:  27
Last checked:         Fri Jan  9 14:13:35 2003
Check interval:       15552000 (6 months)
Next check after:     Wed Jul  9 14:13:35 2003
Reserved blocks uid:  0 (user root)
Reserved blocks gid:  0 (group root)
First inode:          11
Inode size:           128
  Group 0: block bitmap at 3, inode bitmap at 4, inode table at 5
             7928 free blocks, 2033 free inodes, 3 used directories
[output deleted]
##== print the contents of the current inode in directory format
debugfs:  ls -l
   2   40755 (2)      0      0    1024 17-Feb-2003 23:13 .
   2   40755 (2)      0      0    1024 17-Feb-2003 23:13 ..
 4097  40700 (2)      0      0    1024  8-Feb-2003 15:17 lost+found
 8193  40755 (2)      0      0   116736 17-Feb-2003 23:13 dev
57345  40755 (2)      0      0    1024  8-Feb-2003 16:20 var
43010  41755 (2)      0      0    1024 17-Feb-2003 23:13 tmp
47106  40755 (2)      0      0    3072 17-Feb-2003 11:55 etc
[ output deleted ]

##== show inode information for etc
debugfs:  stat etc
Inode: 47106  Type: directory  Mode: 0755  Flags: 0x0  Generation: 461325
User:   0    Group:   0    Size: 3072
File ACL: 0    Directory ACL: 0
Links: 41  Blockcount: 6
Fragment: Address: 0    Number: 0    Size: 0
ctime: 0x3f61d525 -- Fri Feb 17 14:16:05 2003
atime: 0x3f61d4ff -- Fri Feb 17 14:15:27 2003
mtime: 0x3f61d496 -- Fri Feb 17 14:13:42 2003
BLOCKS:
(0):188678, (1):189498, (2):189746
TOTAL: 3

##== chdir into directory inode etc
debugfs:  cd etc
##== show inode information for passwd
debugfs:  stat passwd
Inode: 47121  Type: regular  Mode: 0644  Flags: 0x0  Generation: 461394
User:   0    Group:   0    Size: 1282
File ACL: 0    Directory ACL: 0
Links: 1  Blockcount: 4
Fragment: Address: 0    Number: 0    Size: 0
ctime: 0x3f61d505 -- Fri Feb 17 14:50:04 2003
atime: 0x3f61d4b2 -- Fri Feb 17 23:14:10 2003
mtime: 0x3e5e259c -- Thu Feb 17 14:50:04 2003

```

```

BLOCKS:
(0-1):188692-188693
TOTAL: 2

##== dump the contents of inode that corresponds to passwd
debugfs: cat passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
[ output deleted ]
debugfs: quit

```

Working with OpenBSD images on OpenBSD takes a few more steps. The images need to be configured on the vnode disk device before they can be mounted. An example follows:

### mounting a file system image in OpenBSD:

```

##==
##== associate the image with the vnode pseudo disk device
# vnconfig -v -c svnd0 /var/space/images/2003_02_17_openbsd_attack.bin
svnd0: 7277544448 bytes on /var/space/images/2003_02_17_openbsd_attack.bin
##==
##== mount the image read-only so that the image doesn't change on disk
# mount -o ro /dev/svnd0c /mnt
##==
##== mount the image read-only so that the image doesn't change on disk
# ls -la /mnt
total 9026
drwxr-xr-x  14 root  wheel      512 Nov  4  2002 .
drwxr-xr-x  14 root  wheel      512 Nov  4  2002 ..
-rw-r--r--   2 root  wheel     685 Nov  4  2002 .cshrc
-rw-r--r--   2 root  wheel     179 Nov  4  2002 .profile
drwxr-xr-x   2 root  wheel     512 Oct  4  2002 altroot
drwxr-xr-x   2 root  wheel    1024 Oct  4  2002 bin
-r-xr-xr-x   1 root  wheel   53248 Nov  4  2002 boot
-rw-r--r--   1 root  wheel  4515116 Nov  4  2002 bsd
drwxr-xr-x   4 root  wheel    19968 Sep 12 11:56 dev
drwxr-xr-x  19 root  wheel     2048 Mar 28 12:44 etc
[ output deleted ]
##==
##== unmount the image
# umount /mnt
##==
##== dis-associate the image from the vnode pseudo disk device
# vnconfig -v -u svnd0
svnd0: cleared

```

**fsdb** is the ffs (fast file system) editor in OpenBSD. **fsdb** doesn't have a read-only mode, so it is important to

only work on copies of the image. Also, the 'cd' command in **fsdb** might more appropriately be written as 'ci', as it really means change [active] inode. You can make any inode on the file system the active inode that you are examining by 'cd'ing into it.

### using fsdb on a file system image in OpenBSD:

```
##==
##== associate the image with the vnode pseudo disk device
# vnconfig -vc svnd0 /var/space/images/2003_02_17_openbsd_attack.bin
svnd0: 7277544448 bytes on /var/space/images/2003_02_17_openbsd_attack.bin
##==
##== start fsdb on /dev/svnd0c
# fsdb -f /dev/rsvnd0c
** /dev/rsvnd0c
** File system is already clean
Editing file system `/dev/rsvnd0c'
Last Mounted on /mnt
current inode: directory
I=2 MODE=40755 SIZE=512
    MTIME=Nov  4 19:49:30 2002 [0 nsec]
    CTIME=Nov  4 19:49:30 2002 [0 nsec]
    ATIME=Apr 11 14:06:57 2003 [0 nsec]
OWNER=root GRP=wheel LINKCNT=14 FLAGS=0 BLKCNT=2 GEN=e32f2a77
fsdb (inum: 2)> ls
slot 0 ino 2 reclen 12: directory, `.'
slot 1 ino 2 reclen 12: directory, `..'
slot 2 ino 0 reclen 16: regular, `boot'
slot 3 ino 7488 reclen 16: directory, `altroot'
slot 4 ino 33216 reclen 12: directory, `bin'
slot 5 ino 14016 reclen 12: directory, `dev'
slot 6 ino 42816 reclen 12: directory, `etc'
slot 7 ino 42048 reclen 16: directory, `home'
slot 8 ino 59904 reclen 12: directory, `mnt'
slot 9 ino 6528 reclen 16: directory, `root'
slot 10 ino 5568 reclen 16: directory, `sbin'
slot 11 ino 45888 reclen 16: directory, `stand'
slot 12 ino 27072 reclen 12: directory, `tmp'
slot 13 ino 41472 reclen 12: directory, `usr'
slot 14 ino 6336 reclen 12: directory, `var'
slot 15 ino 6529 reclen 16: regular, `cshrc'
slot 16 ino 6532 reclen 20: regular, `profile'
slot 17 ino 4 reclen 12: symlink, `sys'
slot 18 ino 0 reclen 260: regular, `bsd'
fsdb (inum: 2)> cd etc/passwd
component `passwd': current inode: regular file
I=42860 MODE=100644 SIZE=1033
    MTIME=Feb 27 21:38:23 2003 [0 nsec]
    CTIME=Feb 27 21:38:23 2003 [0 nsec]
    ATIME=Apr 11 13:49:57 2003 [0 nsec]
OWNER=root GRP=wheel LINKCNT=1 FLAGS=0 BLKCNT=4 GEN=4bf628a5
fsdb (inum: 42860)> quit
##==
##== dis-associate the image from the vnode pseudo disk device
# vnconfig -vu svnd0
```

```
svnd0: cleared
```

Like OpenBSD, working with Solaris images on Solaris requires a few extra steps. Solaris has a driver called **lofi** that is a contraction of the words 'loopback file'. Prior to first using **lofiadm**, the kernel won't show the **lofi** module as installed. After the first invocation of **lofiadm**, you should find the **lofi** driver loaded into the kernel. Use the **modinfo** command to display the currently loaded kernel modules. The binary **lofiadm** is included in the **SUNWcsu** package, one of the core packages of the OS. You shouldn't have to install any extra packages to use the **lofi** driver or the utilities associated with it. An example of mounting an image via the **lofi** driver follows:

### mounting a file system image in Solaris:

```
##==
##== register the image available as a block device via the loopback driver:
# lofiadm -a /mnt/images/2003_02_17_attack.bin
/dev/lofi/l
##==
##== verify that the image is registered
# lofiadm
Block Device          File
/dev/lofi/l           /var/space/images/2003_02_17_attack.bin
##==
##== mount the image read-only so that the image doesn't change on disk
# mount -o ro /dev/lofi/l /mnt
##==
##== mount the image read-only so that the image doesn't change on disk
# ls -la /mnt
/mnt:
total 586
drwxr-xr-x  21 root    root          512 Dec  3 04:10 .
drwxr-xr-x  21 root    root          512 Dec  3 04:10 ..
-rw-----   1 root    other        4432 Feb 17 04:25 .sh_history
lrwxrwxrwx   1 root    root           9 Nov 28 06:07 bin -> ./usr/bin
drwxr-xr-x   2 root    nobody        512 Nov 28 07:32 cdrom
drwxr-xr-x  15 root    sys          4096 Feb 17 04:14 dev
drwxr-xr-x   4 root    sys           512 Nov 28 06:29 devices
drwxr-xr-x  41 root    sys          3584 Feb 16 17:00 etc
[ output deleted ]
##==
##== un-mount the image
# umount /mnt
##==
##== unregister the image from the loopback driver
# lofiadm -d /dev/lofi/l
```

**fsdb** is a program that you need to spend some time with prior to getting into a situation where you have to be thinking quickly. Its command syntax is arcane enough that if you master it, you should probably get a medal for perseverance or an award for spending too much time on computers. The most useful documentation that the OS provides is the **fsdb\_ufs(1M)** man page.

## using fsdb on a file system image in Solaris:

```

##==
##== register the image available as a block device via the loopback driver:
# lofiadm -a /mnt/images/2003_02_17_attack.bin
/dev/lofi/1
##==
##== verify that the image is registered:
# lofiadm
Block Device          File
/dev/lofi/1          /mnt/images/2003_02_17_attack.bin
##==
##== browse the image using fsdb:
# fsdb /dev/lofi/1
fsdb of /dev/lofi/1 (Read only) -- last mounted on /
fs_clean is currently set to FSCLEAN
fs_state consistent (fs_clean CAN be trusted)
##==
##== print the super block
/dev/lofi/1 > :sb
      super block:
magic    11954  format  dynamic time    Mon Feb 17 18:36:05 2003
nbfree   605536 ndir    6363   nifree  889612  nffree  8252
ncg      290    ncy1   4631   size   8314960 blocks 8187339
bsize    8192   shift  13     mask   0xfffffe00
fsize    1024   shift  10     mask   0xfffffc00
frag     8      shift  3      fsbtodb 1
cpg      16    bpg    3591   fpg    28728   ipg     3392
minfree  1%     optim  time   maxcontig 16   maxbpg  2048
rotdelay 0ms   fs_id[0] 0x0   fs_id[1] 0x0   rps     120
ntrak   27    nsect  133   npsect  133    spc     3591
trackskew 0    interleave 1
nindir  2048   inopb  64    nspf    2
sblkno  16    cblkno 24    iblkno  32    dblkno  456
sbsize  5120   cgsz   5120   cgoffset 72    cgmask  0xffffffffe0
csaddr  456   cssz   5120   shift   9     mask    0xffffffffe0
cgrotor 187   fmod   0      ronly   0
blocks available in each of 8 rotational positions
cylinder number 0:
[ output deleted ]
##==
##== show current entries in this directory:
/dev/lofi/1 > :ls -l
/:
i#: 2      ./
i#: 2      ../
i#: 2bc0   etc/
i#: 8c02   kernel/
i#: 3      lost+found/
i#: 8c0    usr/
[ output deleted ]
##==
##== set the current block to be examined to block 2bc0 (/etc) and display the
##== information in block 2bc0 as an inode:

```

```

##== note that :pwd will still show the current location as / because you're
##== examining data blocks on the file system. You haven't actually left /.
##== To navigate the directory hierarchy, you need to use :cd <some_path>
/dev/lofi/1 > 2bc0:inode?i
i#: 2bc0          md: d---rwxr-xr-x  uid: 0          gid: 3
ln: 29          bs: 8          sz : c_flags : 0
e00

db#0: 65a8
    accessed: Tue May 27 04:38:06 2003
    modified: Mon May 26 17:00:44 2003
    created  : Tue May 27 04:38:06 2003

/dev/lofi/1 > :ls -l
i#: 2c25          nsswitch.conf
i#: 2c21          passwd
i#: 2c1e          path_to_inst
i#: 2c3e          pwck@
i#: 2bee          rc0@
i#: 6042          rc0.d/
i#: 2bef          rc1@
i#: 6901          rc1.d/
i#: 2bf0          rc2@
i#: 71c2          rc2.d/
i#: 2bf1          rc3@
i#: 7a82          rc3.d/
i#: 2bf2          rc5@
i#: 2bf3          rc6@
i#: 2bf4          rcS@
i#: 834e          rcS.d/
i#: 2c2d          shadow
i#: 2c27          syslog.conf

[ output deleted ]

##==
##== set the current block to be examined to block 2c21 (/etc/passwd) and display
the
##== information in block 2c21 as an inode:
/dev/lofi/1 > 2c21:inode?i
i#: 2c21          md: ----r--r--r--  uid: 0          gid: 3
ln: 1          bs: 2          sz : c_flags : 0
20f

db#0: 6554
    accessed: Tue May 27 04:37:58 2003
    modified: Thu Nov 28 08:18:06 2002
    created  : Tue May 27 04:37:58 2003

##==
##== display the information in current block as ASCII data:
##== you can display the block in hex using: 0:db:block,*/X
/dev/lofi/1 > 0:db:block,*/c
1955000:          r   o   o   t   :   x   :   0   :   1   :   S   u   p   e   r
1955010:          -   U   s   e   r   :   /   :   /   s   b   i   n   /   s   h
1955020:          \n  d   a   e   m   o   n   :   x   :   1   :   1   :   :   /
1955030:          :   \n  b   i   n   :   x   :   2   :   2   :   :   /   u   s
1955040:          r   /   b   i   n   :   \n  s   y   s   :   x   :   3   :   3
1955050:          :   :   /   :   \n  a   d   m   :   x   :   4   :   4   :   A

```

```

1955060:    d   m   i   n   :   /   v   a   r   /   a   d   m   :   \n   l
1955070:    p   :   x   :   7   1   :   8   :   L   i   n   e   P   r
1955080:    i   n   t   e   r   A   d   m   i   n   :   /   u   s   r
1955090:    /   s   p   o   o   l   /   l   p   :   \n   u   u   c   p   :
19550a0:    x   :   5   :   5   :   u   u   c   p   A   d   m   i   n
19550b0:    :   /   u   s   r   /   l   i   b   /   u   u   c   p   :   \n
19550c0:    n   u   u   c   p   :   x   :   9   :   9   :   u   u   c   p
19550d0:    A   d   m   i   n   :   /   v   a   r   /   s   p   o   o
19550e0:    l   /   u   u   c   p   p   u   b   l   i   c   :   /   u   s
19550f0:    r   /   l   i   b   /   u   u   c   p   /   u   u   c   i   c
1955100:    o   \n   s   m   m   s   p   :   x   :   2   5   :   2   5   :
1955110:    S   e   n   d   M   a   i   l   M   e   s   s   a   g   e
1955120:    S   u   b   m   i   s   s   i   o   n   P   r   o   g
1955130:    r   a   m   :   /   :   \n   l   i   s   t   e   n   :   x   :
1955140:    3   7   :   4   :   N   e   t   w   o   r   k   A   d   m
1955150:    i   n   :   /   u   s   r   /   n   e   t   /   n   l   s   :
1955160:    \n   n   o   b   o   d   y   :   x   :   6   0   0   0   1   :
1955170:    6   0   0   0   1   :   N   o   b   o   d   y   :   /   :   \n
1955180:    n   o   a   c   c   e   s   s   :   x   :   6   0   0   0   2
1955190:    :   6   0   0   0   2   :   N   o   A   c   c   e   s   s
19551a0:    U   s   e   r   :   /   :   \n   n   o   b   o   d   y   4
19551b0:    :   x   :   6   5   5   3   4   :   6   5   5   3   4   :   S
19551c0:    u   n   O   S   4   .   x   N   o   b   o   d   y   :
19551d0:    /   :   \n   k   r   h   :   x   :   1   1   1   9   :   1   1
19551e0:    1   9   :   K   r   4   D   H   a   X   0   R   y   o
19551f0:    :   /   e   x   p   o   r   t   /   h   o   m   e   /   k   r
1955200:    h   :   /   u   s   r   /   b   i   n   /   k   s   h   \n
[ output deleted ]
##==
##== unregister the image from the loopback driver:
# lofiadm -d /dev/lofi/1

```

Sometimes it is necessary to see the information that is contained in the image without any interpretation by file system debuggers. Several tools can be used for this purpose.

A hex-editor, such as **hexedit**, is useful for browsing file system images without the constraints of a file system debugger. **hexedit** is available as part of the RedHat distribution and can be installed on OpenBSD using the ports tree in `/usr/ports[1]`.

When a hex-editor isn't available, you can use **emacs** or **vim**. To use emacs to view a file system image, start emacs using **emacs <image\_name>**. Turn on read-only mode using 'Esc-x toggle-read-only'. Next, changed to the hex-edit view using 'Esc-x hexl-mode'. To exit, use 'Ctrl-x Ctrl-c'. For more information on how to us **emacs**, type 'Ctrl-h t' to start the **emacs** tutorial.

**vim**'s hex-editing support is a bit more klunky than emacs'. To start **vim** in read-only binary mode with no swap file, run **vim -nRb <image\_name>**. Next type 'Esc:%!xxd' to convert the file to being displayed in a hex format. To exit, type 'Esc:q!'. For a tutorial on using **vim**, type **vimtutor** from the command line. For more help with **vim** hit the F1 key while in **vim**.

The amount of memory that RAM and swap provide is a practical limitation for using editors to view large files. Once the machine starts swapping (swapping occurs when RAM is used up and swap starts to be used), your session can slow down significantly. Use **split** or **csplit** to divide the file system image into multiple parts when you run into this problem. To reiterate: you need to be using a copy of the image when doing anything that will or has the potential to change your image. Even if you've specified options that make your tools to operate in read-only mode, you shouldn't be working on the only copy you have. If you are in a situation where you're gathering evidence for an investigation and the defense can question your methodology because you might have tainted the evidence by writing to it, you can lose your case. One should also be cognizant of the changes that are made as you manipulate the copy. The more you change it (split it, compress it, fiddle around in an editor), the more you have to be careful about the conclusions you're drawing. Check, check, and re-check. When you're done, re-assemble the copy and compare the hash of the copy to the hash of the original. This will help you verify that your transforms didn't introduce any changes.

When a hex editor, **emacs**, and **vim** aren't available, you can use **od**, along with a pager such as **less**, **more** or, on Solaris, **pg**. **od** is a tool that is present on OpenBSD, RedHat Linux, and Solaris. **od** dumps data in various representations. Using a pager such as **less** is recommended. **more** is ok, too, but **less** has a fuller feature set and is more efficient at dealing with the amount of data that can be generated when **od** dumps a file system image. You might say that **less** is more than **more**. **od -vxca** dumps in hex, and displays characters with C style escapes, their ASCII equivalents, and prints duplicate lines. You get three rows of dump data for every 32 (0x20) bytes of data that **od** is processing. **od** on OpenBSD and RedHat displays the dump in **big-endian** order (the highest byte has the lowest address) and **od** on Solaris displays the dump in **little-endian** order (the lowest byte has the lowest address).

## Transferring data from the compromised host

If you need to transfer data off a host that is on-line prior to a creating a forensic image, there are a couple of issues to consider.

- The transfer should perturb the file system as little as possible
- The transfer should ideally occur over a secure channel to a trusted host
- The tools you use should not be compromised. Running them from CD is ideal
- If the investigation could involve law enforcement, you should be following procedures that will most likely be honored as valid by investigators that you might have to work with
- If you're using credentials to authenticate to a remote host for storing data, is it possible that an attacker who is logged into the machine can use or steal your credentials?

There are a number of ways of saving the most important data. The first covered is to use a hub or crossover cable to transport data from the compromised host to a second host (such as a laptop). Using ssh and using several tools together to create move data via TLS/SSL is also discussed.

Connecting the compromised host to a private network that has a host on which the forensic data can be stored can provide a secure medium with high bandwidth. This method has some drawbacks, however:

- Connections to remote hosts begin to timeout out or are dropped once the host is disconnected.
- Observation of the attacker's behavior is difficult as the attacker is no longer able to communicate with the machine.

- Observation of some of the programs that the attacker might have installed might be difficult. If the attacker has installed a client that needs to communicate with a remote server, it becomes difficult to observe the client-server interaction when the client can no longer communicate with the server.
- Figuring out how the attacker is connecting to the machine might be difficult because the machine can no longer be contacted by the attacker.

As always, this is yet another trade off where some thinking and planning needs to be done prior to making a decision.

1. set up the physical layer.
  - connect the NICs with a crossover cable that you buy or make[14].
  - connect both NICs into a hub.
2. set up the forensics data storage host on the same IP network as the compromised host.
3. send data to the forensics data storage host using netcat[15] and dd.

#### moving data with dd and netcat:

```
sechost# nc -lp 8091 > <save_file>
##== use a tcp over ethernet friendly block size of 1460
hckdhost# dd if=/dev/hda bs=1460|nc -w10 <sechost> 8091
sechost# od -xvca <save_file>|less
```

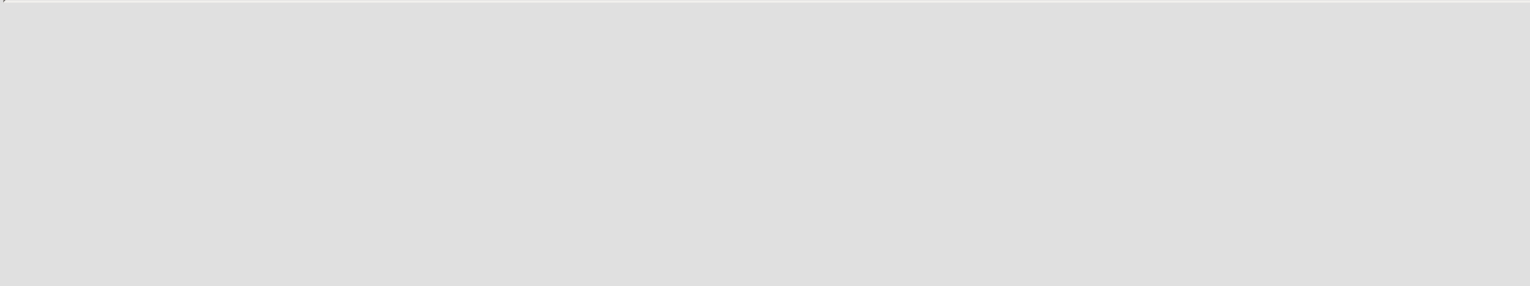
Another tool called **socat**[16] can be used and removes the need for using **dd**. **socat** is significantly more flexible than netcat because it can use a variety of methods to move data. Note the ignoreeof option used in each of the commands below that makes it possible to create an image of /dev/hda.

#### moving data via socat:

```
##== use a tcp over ethernet friendly block size of 1460
sechost# /usr/local/socat/bin/socat -b 1460 -t5 OPEN:/dev/hda,ignoreeof,rdont TCP4:
<sechost>:8091
hckdhost# /usr/local/socat/bin/socat -b 1460 -t5 TCP4-LISTEN:8091 OPEN:>save_file<,
create,excl,largefile,ignoreeof
sechost# od -xvca <save_file>|less
```

If you can't or don't want to pull the machine off the network, you can use ssh to securely transport data off the host. The following **script** saves current network connection state, the current process list, various system information, and a file system image to a remote host.

#### example script for moving data to a secure host via ssh:



```
#!/bin/bash

# This is a quick hack for demonstration purpose only. It needs
# to be adapted to your environment. This script works on Linux.
# YMMV elsewhere.

rem_host="192.168.27.23"

ssh="ssh forensics@${rem_host}"

# get uname -a, uptime, and Debian or RedHat version info
echo -e `uname -a` "\n" `uptime` "\n"
  `[ -s /etc/debian_version ] &&
    echo Debian $(cat /etc/debian_version) || cat /etc/redhat-release`
  | ${ssh} "dd of=/var/tmp/incidents/sysinfo"

# save process information
ps auwx | ${ssh} "dd of=/var/tmp/incidents/processes_bsd"
ps -eflyc | ${ssh} "dd of=/var/tmp/incidents/processes_sysv"

# save list of open files
lsof | ${ssh} "dd of=/var/tmp/incidents/lsof"

# save networking information
netstat -A INET -anv | ${ssh} "dd of=/var/tmp/incidents/netstat_infos"
lsof -Pni | ${ssh} "dd of=/var/tmp/incidents/lsofnet_infos"

# save loaded modules
# use modinfo on Solaris, modstat on OpenBSD
lsmod | ${ssh} "dd of=/var/tmp/incidents/modules"

# wtmp info from last
# snagging /var/log/[wu]tmp* might not be bad idea
last | ${ssh} "dd of=/var/tmp/incidents/last"

# info from process accounting
# snagging /var/account/* might not be a bad idea
# lastcomm is used here, dump-acct can work too
# system accounting (sar) if enable can be useful too. files are usually in
# /var/log/sysstat
lastcomm | ${ssh} "dd of=/var/tmp/incidents/last"

# save /etc and logs
tar cvjf - /etc /var/log/* | ${ssh} "dd of=/var/tmp/incidents/files_and_logs.tar.bz2"

# use mount to determine currently mounted drives to image. This might need
# tweaking depending on your system, so that it only picks up the drives you want.
# Is compressing with bzip2 ok for forensics in your world? Since you're applying
# a transform before taking an md5sum, it's possible it could cause an issue.
Consult
# LE or Legal.
#
# This doesn't handle swap if swap lives on a drive that isn't in the list that
# mount generates. Use 'swapon -s'.
```

```
#
#  bzip2 can run on whichever machine is faster, or it can be used before the
#  data goes over the network. If your network is fast enough, bziping on the remote
#  host is a good idea to conserve space. Software compression can take more time
than
#  it takes to move uncompressed data across a network, if the network is fast
enough.
#  In these situations compress only if you're worried about space, or compress
after
#  the transfer is done.
for drive in `mount |grep '^\/dev.* (rw'|awk '{print $1}'|sed 's/[0-9]\+$//'|sort|
uniq`
do
    drive_name=`basename ${drive}`
    dd if=${drive} |${ssh} "bzip2 -9c|dd of=/var/tmp/incidents/${drive_name}.raw.bz2"
done
```

Another way of propagating information safely is to use **socat**[\[16\]](#). **socat** currently doesn't support receiving TLS/SSL connections, so **stunnel**[\[17\]](#) is used to provide that functionality on the remote host.

The following steps show how to move data using **socat** and **stunnel**:

1. Generate a self signed certificate[\[18\]](#).
2. Append diffe-hellman parameters to the self signed certificate.
3. Make a symlink to the cert with the cert's hash.
4. Edit the hosts.allow on the remote host so that appropriate machines can connect to it.
5. Start up stunnel telling it to exec frecv.pl for each incoming connection.
6. Send files as needed using socat.

#### moving data using socat and stunnel:

```
##== 1
sechost# openssl req -new -x509 -nodes -days 365 -out stunnel.pem -keyout stunnel.pem
##== 2
sechost# dd if=/dev/urandom count=8 |openssl dhparam 512 >> stunnel.pem
##== 3
sechost# ln -sf stunnel.pem `openssl x509 -noout -hash < stunnel.pem`.0
##== 4
sechost# vi /etc/hosts.allow
##== 5
sechost# stunnel -fd <sechost>:9001 -l ./frecv.pl -p stunnel.pem -P none
##== use a tcp over ethernet friendly block size of
```