SANS/GIAC Certified Forensic Analyst (GCFA) Practical Assignment version 1.0

SANS GIAC

Chris Calabrese
September 2002

Table of Contents

Part 1 - OPTION 2: Perform a Forensic Tool Validation

Scope of Test

This test concerns itself with GNU **`tar`** and GNU **`cpio`**, which are tools for copying, archiving, and un-archiving data files.

Specifically, the test will examine whether these tools are forensically sound when used in various common modes, and whether using their **`--sparse`** features can reduce disk space and real time requirements of a forensic investigation.

Tool Description

Products Tested

The products tested are GNU **`tar`** and GNU **`cpio`**, both of which are data archiving tools for Unix-like systems.

My originally intention was to test only GNU **tar** version 1.13.19 and GNU **cpio** version 2.4.2, which are included with the Red Hat Linux version 7.1 distribution installed on my test system. However, it soon became evident that these versions did not perform as expected.

After discussing the issues with the GNU tool maintainers[1] I eventually included GNU **tar** version 1.13.25 from ftp://alpha.gnu.org/gnu/tar/ and GNU **cpio** version 2.5 from ftp://ftp.gnu.org/gnu/cpio/ in my test.

What the Tools are Supposed to Do

These tools are used for copying data files on Unix-like systems (though they've also been ported to Windows). This includes copying data in and out of file and/or tape archives, such as when archiving several files into a single archive file for organization and distribution.

In this way, **tar** and **cpio** are very similar to WinZip. The main differences being that **tar** and **cpio** are easier to script, and they only do the archiving part, not the compression part (usually done in the Unix world these days with **gzip**/**gunzip**).

Another difference between **tar**/**cpio** vs. WinZip is actually a difference of Unix vs. MS Windows. That is the ability to read not just actual data files through the filesystem, but also other constructs of the operating system through device files and pseudo-filesystems. For example, you can take a snapshot of OS information on process status by accessing the **/proc** pseudo-filesystem/directory through **tar**/**cpio**.

You might also think that **tar**/**cpio** can be used with Unix device files to directly access raw filesystem images (e.g., **/dev/root** on Linux to access the root filesystem) or the system memory map (e.g., **/dev/mem** and **/dev/kmem**), including unallocated space. Unfortunately, they cannot do this directly since they backup device file metadata rather than the contents. Instead, you will need another tool that can image device files (**dd** being the obvious choice) and can then point **tar**/**cpio** at the output of that tool.

Differences between **tar** and **cpio** themselves are:

- They use different internal representations for their archive files.
- **tar** is easier to use when archiving a specific set of named files such as in
  ```
  tar xf myarchive.tar fileA fileB fileC
  ```
- **cpio** is more powerful in being able to archive a large set of files by specific parameters, such as in:
  ```
  find . –name '*.sql' | cpio –o > myarchive.cpio
  ```

As a result, people generally learn **tar** first and are more familiar with it. But, since **cpio** is more powerful, a forensic investigator should be familiar with both.

How This Helps the Forensic Investigator

The forensic investigator can use these tools to create data archives as part their evidence gathering. They would use such archives for pristine storage of system images, distribution of "working images," etc. The tools can later be used to un-archive the data back to its constituent files in a forensically sound way (i.e., it can be shown that the contents of the re-constituted files will be identical to the contents of the original files).

For example, an analyst might use **`tar`** or **`cpio`** to read evidence data (individual files, while filesystems, or OS-level process information) from of a machine under examination and then burn the resulting data archive into a CD-ROM. They would then use this original CD-ROM as a master copy of the data for use as courtroom evidence, and to make "working copies" used to un-archive the data onto a forensic workstation for analysis.

This ability to create forensically sound copies of the data from a system under investigation is important to the forensic investigator because it allows the investigator to "play with" the data without the risk of modifying the master copy of the data and thus calling into question the quality of any evidence gathered.

Aside from these fundamental properties of archiving/un-archiving data, these tools have another interesting feature - their ability to create Unix/Linux filesystem "holes" when called with the "**`–sparse`**" command-line option. Filesystem holes are a feature of Unix/Linux systems whereby they synthesize zero-bit-filled disk blocks without actually taking up disk space. By using this feature, the forensic investigator can save significant amounts of disk space when analyzing disk images, which typically contain large amounts of unused (i.e., zero-filled) space.

This is important to the forensic investigator because it allows the examination of very large (but sparse) data sets on a forensic workstation with a relatively small amount of disk space. For example, during the SANS Track 3 courses at SANS 2002, I was able to analyze over 6GB of filesystem data from a compromised system, even though my workstation had less than 2GB of disk space available.

Additional System Files

System Libraries for GNU `tar` version 1.13.19 (bundled with Red Hat Linux version 7.1):

System Libraries for GNU `cpio` version 2.4.2 (bundled with Red Hat Linux version 7.1):

System Libraries for GNU `tar` version 1.13.25 from ftp://alpha.gnu.org/gnu/tar/:

LDD alpha version of tar, screen 1

LDD alpha version of tar, screen 2

System Libraries for GNU `cpio` version 2.5 from ftp://ftp.gnu.org/gnu/cpio/:

Other System Files:

Other system files (not shared libraries) were determined using **`lsof`** during the execution of the programs, as shown in the Data and Results section.

Running from a CD-ROM

These tools can run directly from CD-ROM without installation on the system. In fact, the SANS System Forensics, Investigations, and Response Course CD v1.2 (distributed in my Track 8 class sessions at SANS 2002) includes binaries for **`tar`** that can be run off of the CD.

Interestingly, however, while the CD includes Solaris and MS Windows versions, it does not include Linux versions, even though Linux is the primary OS for the Track 8 laboratories...

Static Compilation

It is possible to compile these tools statically (i.e., such that they do not rely on the shared-libraries/DLL's of the system they are running on). In this way, they can be used in an evidentiary sound way (i.e., so that the shared-libraries/DLL's installed on the system they're running on do not affect the ability of the tools to accurately recreate their input data as it passes through archive/un-archive cycles).

The analyst can still use non-static compilations in an evidentiary sound way, but only if the analyst trusts the shared library files on the system the process is running on. This is often the case when un-archiving collected data onto a trusted analysis machine.

Libraries on an untrusted machine could be verified using HMAC signatures (e.g., with **`md5sum`**). However, this requires a trusted and statically compiled signature checker...

Test Methodology

Test Apparatus

The test environment hardware is a Compaq Armada M700 laptop with 512 MB of RAM. The OS is a stock install of Red Hat Linux version 7.1 as a guest OS under VMWare Workstation version 3.0.0 (with a stock install of Windows XP Professional with all patches from Windows Update as the host OS). I allocated the VMWare partition 192MB of RAM.

Environmental Conditions

I performed all testing using a networking configuration I believe is extremely secure – The VMWare partition was configured without any network connectivity (all data transfers done via floppy disks accessed by both the Linux VMWare partition and the native Windows XP OS), while the host Windows XP system was configured with a very tight local firewall allowing only outbound connections. The laptop was physically located in my office in Franklin Lakes, NJ during testing. At other times, it was located in my car, or in various rooms of my house in Montclair, NJ. The important point is that I had physical control over it all times.

Data Sources

All data used in testing came from the **`honeypot.hda.8`** file of **`ForensicChallenge/challenge/images.tar`** on the SANS Systems Forensics, Investigations, and Response Course CD v1.2.

Description of the Procedures

Equipment identification:

Equipment identified via visual inspection. Equipment prepared by loading Red Hat 7.1 via FTP download, with VMWare partition created after Red Hat loaded (on separate disk partition). As mentioned in the Test Apparatus section, the test environment hardware is a Compaq Armada M700 laptop with 512 MB of RAM. The OS is a stock install of Red Hat Linux version 7.1 running as a guest under VMWare Workstation version 3.0.0 (with Windows XP as the host OS), and I allocated VMWare partition 192MB of RAM.

Checks before testing

Hardware checked through normal usage (i.e., this laptop is my normal desktop system and I use it constantly).

Initial OS installation ensured by loading OS from known good sources (i.e., official Red Hat mirror).

OS installation initially checked by using this setup during the Track-8 sessions at SANS 2002, where it performed properly in all laboratories.

Continued correctness of OS installation assured through the measures described in the Environmental Conditions section.

Final check before these tests were performed included booting up the Red Hat / VMWare partition and checking to make sure it could still access all hardware peripherals properly.

How documentation will be kept

All permanent documentation included directly in this report. Temporary files stored in **`/home/p644cc/gcfa`** on the Linux partition of the laptop.

Procedures to protect the integrity of test results

- Configure VMWare so that the Linux image has no network connectivity.
- Physically secure Laptop.

Procedures to protect repeatability/reproducibility of test results

- Configure VMWare so that the Linux image has no network connectivity.
- Minimize extraneous jobs running on the test system during testing.
- Run each test involving timing multiple times
- Make no hardware changes to the system during the testing period
- Make no OS changes to the host Windows XP system or to the Linux VMWare system during the testing period

Procedures to protect integrity of test results

- Test results recorded into this document in real time (i.e., as they were obtained).
- Document itself stored on laptop, which I had physical control over and which has very tight local firewall to reduce the possibility of network intrusion (only allows outgoing connections).

- No individuals allowed access to laptop beside the author (e.g., corporate desktop support team does not know administrator password).

Criteria for Approval

Expected results:

1. The tools do not modify data as it passes through archive/unarchive cycles.
2. The tools use significantly less physical disk space when called with the `--sparse` option. This is important to the forensic investigator because it allows the examination of very large (but sparse) data sets on a forensic workstation with a relatively small amount of disk space. See the How This Helps the Forensic Investigator section for more information on how this works and what its benefits are.
3. Running times of forensic analysis routines (`strings`, `unrm`, `find`) are faster on files created by the tools being tested when there are called with the `--sparse` option (as less physical disk accesses are necessary).
4. Running times to archive/unarchive files are faster when the tools are called with the `--sparse` option (as less physical disk accesses are necessary)

Use of system files:

Determine shared libraries with '`ldd -v`' as shown in the Additional System Files section.

Confirm and detect other (non shared-library) system files by executing `lsof` while tools are executing. Note that I later realized that running `strace` would have been superior, but `lsof` produces reasonable results given that library files were already determined with `ldd`.

Detection of evidence manipulation:

Use MD5 signatures (i.e., `md5sum`) to detect evidence manipulation. Exact command lines are given below in the Execution: section.

Detection of file sizes

File sized determined using `df` after using `du` to determine that the bundled version of `df` on Red Hat Linux version 7.1 does not report disk space used by filesystem "holes."

Detection of running times

Running times detected using the `time` command. Exact command lines are given below in the Execution: section.

The astute reader may notice that **time** output is in two slightly different formats. It appears that the GNU Bourne Again Shell (**bash**) bundled with my Red Hat 7.1 installation calls its own built-in **time** function only when **time** is the first command in a pipeline, falling through to **/bin/time** at other times. In contrast, the Korn Shell (**ksh**) always calls its built-in **time** function as long as **time** is the first command of a pipe segment. By way of example, both **bash** and **ksh** would call their built-in **time** function for

```
time wc -l fubar
```

However, only **ksh** calls its built-in **time** function for

```
cat fubar | time wc -l
```

Execution:

Testing was executed as follows:

1. Extract image files from CD using
   ```
   mkdir base; cd base;
   tar xf  /mnt/cdrom/ForensicChallenge/challenge/images.tar
   ```
   to unarchive files from the CD to local disk without using the sparse option. Unzip **honeypot.hda8.dd.gz**, which will become the basis for future tests. Record external library files used; file sizes, compare files to expected MD5 signatures.
2. Create a **tar** archive of the **honeypot.hda8.dd** file with
   ```
   mkdir ../tmp; time tar -cf ../tmp/tar *.dd
   ```
   Record external library files used; file sizes, and time for three runs.
3. Unarchive the archive created in the previous step with
   ```
   cd ../tmp; time tar -xf tar
   ```
   Record external library files used; record file sizes; check MD5 signatures, and record time for three runs.
4. Mount the image with
   ```
   mkdir mountpoint;
   mount -ro,loop,nodev,noexec,noatime $PWD/*.dd $PWD/mountpoint
   ```
   and record running times for three runs on each of forensic analysis tools on the **tmp/honeypot.hda8.dd** and its mounted image with
   ```
   time strings honeypot.hda8.dd > /dev/null
   time unrm honeypot.hda8.dd > /dev/null
   time find mountpoint > /dev/null
   ```
5. Unmount **mountpoint** and remove no-longer-needed files in **tmp** directory with
   ```
   umount $PWD/mountpoint
   cd ../base
   rm -rf ../tmp
   ```
6. Repeat steps 2-5, but using the **--sparse** option to all invocations of **tar**.
7. Repeat steps 2-5, but using both **–sparse** and GNU **tar** 1.13.25 from ftp://alpha.gnu.org/gnu/tar/ rather than the system-supplied version of **tar**.
8. Copy the **honeypot.hda8.dd** file with
   ```
   cd ../base; echo *.dd | time cpio --sparse -p ../tmp
   ```
   Record external library files used; file sizes, and time for three runs. Separate archive creation/explosion steps won't work here because GNU **cpio** does not implement **--sparse** for un-archiving.

9. Repeat steps 4-5 to mount the image, record running times, unmount the image, and remote the no-longer needed **`tmp`** directory, but for **`cpio`** rather than **`tar`**.
10. Repeat steps 8-9 using GNU **`cpio`** 2.5 from ftp://ftp.gnu.org/gnu/cpio/ rather than the system-supplied version of **`cpio`**.

Test Results

Data and Results

Raw data:

Raw Test Results

Action	Running Times	Disk Space Usage for **`hda8`**	External Files/Libraries Used	Files Manipulated?
Unarchive from CD using **`/bin/tar`** without sparse option into **`base`** directory	NA	261 MB	`/lib/ld-2.2.2.so` `/lib/i686/libc-2.2.2.so` `/lib/libnss_files-2.2.2.so` `/usr/lib/locale/EN_US/*` See also the corresponding screen shot in the Additional System Files section	No
Create archive using **`/bin/ar`**	1m41.105s,0m0.480s,0m12.240s 1m55.987s,0m0.400s,0m10.450s 1m47.595s,0m0.520s,0m15.200s **Avg real time 108.229s**	261 MB	As above	NA
Unarchive using **`/bin/tar`**	1m43.874,0m0390s,0m014.980s 1m43.208s,0m0360s,0m11.720s 1m42.247s,0m0.330s,0m10.250s **Ave real time 103.110s**	261 MB	As above	No
Forensic tools run on image created with **`/bin/tar`**	**`strings`** 1m58.401s,1m37.890s,0m6.870s 1m50.488s,1m37.890s,0m7.140s 1m51.464s,1m37.340s,0m7.620s **`unrm`** 0m41.479s,0m1.260s,0m5.640s 0m41.326s,0m1.350s,0m4.590s 0m44.216s,0m1.020s,0m4.730s **`find`** 0m1.523s,0m0.020s,0m0.160s	NA	NA	NA

Raw Test Results				
Action	**Running Times**	**Disk Space Usage for `hda8`**	**External Files/Libraries Used**	**Files Manipulated?**
	0m0.093s,0m0.000s,0m0.090s 0m0.165s,0m0.040s,0m0.040s **Total real time 438.131s**			
Create archive using **`/bin/tar --sparse`**	1m35.229s,0m0.470s,0m16.150s 1m44.012s,0m0.420s,0m13.560s 1m45.111s,0m0.380s,0m11.030s **Ave real time 101.451s**	**261 MB - no savings**	As above for `tar` without **`--sparse`**	NA
Unarchive using **`/bin/tar --sparse`**	1m41.621s,0m0.290s,0m10.690s 1m40.360s,0m0.320s,0m9.760s 1m47.134s,0m0.380s,0m9.280s **Ave real time 103.038s**	**261 MB - no savings**	As above	No
Forensic tools run on image created with **`/bin/tar --sparse`**	**`strings`** 1m54.930s,1m37.450s,0m6.920s 1m55.260s,1m37.960s,0m8.640s 1m51.697s,1m37.650s,0m7.950s **`unrm`** 0m46.725s,0m1.080s,0m4.290s 0m43.872s,0m1.050s,0m4.190s 0m41.342s,0m0.990s,0m3.620s **`find`** 0m1.798s,0m0.070s,0m0.190s 0m0.250s,0m0.010s,0m0.030s 0m0.097s,0m0.040s,0m0.050s **Total real time 475.971s**	NA	NA	NA
Copy file using **`/bin/cpio --sparse`**	0:59.92,4.92,8.39 0:58.38,4.6,8.75 1:00.53,5.25,10.46 **Avg real time 59.61s**	51 MB	As above for `tar`	**YES - apparently it strips trailing NULL's**
Forensic tools run on image created with **`/bin/cpio --sparse`**	**`strings`** 1m48.760s,1m35.690s,0m7.300s 1m54.440s,1m35.570s,0m7.580s 1m46.540s,1m34.860s,0m7.390s **`unrm`** 0m11.902s,0m1.080s,0m5.550s 0m11.132s,0m1.130s,0m5.150s 0m12.000s,0m1.080s,0m4.880s **`find`** 0m1.412s,0m0.050s,0m0.150s, 0m0.100s,0m0.040s,0m0.050s 0m0.094s,0m0.030s,0m0.050s	NA	NA	NA

Raw Test Results				
Action	**Running Times**	**Disk Space Usage for `hda8`**	**External Files/Libraries Used**	**Files Manipulated?**
	Total real time 366.378s			

Create archive using `tar-1.13.25 --sparse`	1m34.493s,0m0.550s,0m10.930s
	1m40.214s,0m0.640s,0m10.300s
	1m41.372s,0m0.290s,0m8.150s
	Avg. real time 98.69s
		261 MB - no savings	`/lib/ld-2.2.2.so`
			`/lib/librt-2.2.2.so`
			`/lib/i686/libc-2.2.2.so`
			`/lib/i686/libpthread-0.9.so`
			`/lib/libnss-files-2.2.2.so`
			`/usr/lib/locale/EN_US/*`
			
			See also the corresponding screen shot in the Additional System Files section
				NA

Unarchive using `tar-1.13.25 --sparse`	1m38.387s,0m0.270s,0m7.060s Did not continue - no advantage over older versions	**261 MB - no savings**	As above	No
Forensic tools run on image created with `tar-1.13.25 --sparse`	Not performed	NA	NA	NA
Copy file with `cpio-2.5 --sparse`	48.82,12.12,7.53 47.57,8.97,6.81 52.02,9.11,7.10 **Avg. real time 49.47**	51 MB	As above for other `cpio` runs	No (unlike the older version of **cpio**!)
Forensic tools run on image created with `cpio-2.5 --sparse`	**`strings`** 1m20.011s,1m9.190s,0m7.200s 1m24.234s,1m6.610s,0m7.020s 1m15.162s,1m4.620s,0m6.050s **`unrm`** 0m9.185s,0m0.760s,0m4.540s 0m9.487s,0m0.760s,0m5.030s 0m8.859s,00.820s,0m4.580s **`find`** 0m1.452s,0m0.020s,0m0.170s 0m0.087s,0m0.020s,0m0.050s	NA	NA	NA

Raw Test Results				
Action	**Running Times**	**Disk Space Usage for `hda8`**	**External Files/Libraries Used**	**Files Manipulated?**
	0m0.055s,0m0.030s,0m0.030s **Total real time 268.532s**			

Note that I decided not to include screen shots in this section because I felt it would make the section more difficult to follow given the large amount of information being presented.

Comparison to expected results:

- Neither version of GNU **`tar`** works as expected with **`--sparse`**. This has been reported to the GNU **`tar`** tool maintainers[1]. This is really too bad because **`tar`** is much more popular than **`cpio`**.
- GNU **`cpio`** version 2.4.2 (bundled with my Red Hat Linux version 7.1 installation) does not preserve data with **`--sparse`**. I suspected that this is because files ending in zero-filled blocks never have the final **`write`**() needed to force the filesystem to recognize the end of the "hole." The GNU **`cpio`** tool maintainers confirmed this suspicion in private correspondence[1].
- GNU **`cpio`** version 2.5 works as expected with **`--sparse`**, and exhibits the expected savings in disk space and running times (roughly an 80% savings in disk space and a 40% savings in running times in the test scenario). According to the tool maintainers, the actual fix was to add a final

  ```
   lseek(fd, -1, SEEK_CUR); write(fd, "", 1);
  ```

  when a file ends in zero-bytes. This fix originated from Debian Linux[1].

Analysis

As discussed above in the How This Helps the Forensic Investigator section, the primary use of these tools by a forensic investigator is not to directly produce evidence for analysis, but to facilitate evidence gathering and preservation. Thus, the most important analysis that can a forensic investigator can perform on the data obtained using these tools is whether the data files stored in archives created with these tools can be reconstituted to their original form.

Looking back at the Data and Results, the answer is a qualified yes. Qualified because the GNU **`cpio`** version 2.4.2 bundled with my Red Hat Linux version 7.1 installation does not always preserve data in all cases when used with **`--sparse`**. However, Forensic investigators can deal with this by not using **`--sparse`** with files ending in zero-bytes, or by moving to GNU **`cpio`** version 2.5.

The forensic investigator can easily verify this for themselves by checking HMAC signatures on the original files against those derived from the archive (e.g., with **`md5sum`**). They can also verify that any copies of the archive are correct by checking signatures on the original archive file against the new copy.

Presentation

Since the output of these tools is not evidence directly, but a preservation of evidence, what might be presented by a forensic investigator about the output of these tools for others to interpret would not be the archive files themselves, but rather evidence that the archive files can correctly re-constitute the original evidence files they were built from. In fact, this is the very analysis of the previous section.

An easy way to do this is to show screen dumps of an HMAC signature checker run against the original evidence files and then against files derived from the archives. There is an example of such a screen shot in Part 2 of this document.

Even better would be to show screen dumps for the original evidence files, and then un-archive the files and run the signature checker against them live. You'd need a laptop with you for this, but it would be worth it so you can show exactly how HMAC signature checking works.

Conclusion

`tar` and **`cpio`** are extremely useful tools to the forensic investigator because of their ability to preserve evidence on Unix systems. Not only can they preserve individual files, but also whole filesystems, including unallocated filesystem space, through device files. And, they can preserve OS process status through **`/proc`** (available on most modern Unix-like systems).

Further, **`cpio --sparse`** can be extremely useful to the forensic investigator in reducing the disk space and time needed to perform a forensic analysis - as long as you make sure you have GNU **`cpio`** version 2.5!

However, a working **`tar --sparse`** , **`dd --sparse`**, and **`gunzip --sparse`** would be even better.

Additional Information

- GNU web page for **`tar`** - http://www.gnu.org/manual/tar/index.html
- GNU **`tar`** maintainer - tar-bugs@gnu.org
- GNU web page for **`cpio`** - http://www.gnu.org/software/cpio/cpio.html

- GNU **cpio** maintainer - bug-cpio@gnu.org
- Red Hat main web page - http://www.redhat.com/

References

1. Private e-mail correspondence with the GNU **tar** and GNU **cpio** tool maintainers (tar-bugs@gnu.org and bug-cpio@gnu.org respectively), June and July 2002.

Part 2 - Analyze an Unknown Binary

Binary Details

File Name	`Sn.dat`
MAC Times	Modified 2002 Apr 11 09:29:58 Other times not evident with MS-DOS ZIP archive type, as the FAT filesystem that the archive was created on does not have this concept. I verified that I could get other MAC times from the version of **`zipinfo`** I used by creating/examining Unix ZIP archive, which worked as expected.
File owners	Again, not evident with MS-DOS ZIP archive type, as the FAT filesystem that the archive was created on does not have this concept. Again, I verified that the version of **`zipinfo`** I used could determine this information by creating/examining a Unix ZIP archive, which worked as expected.
File size	399,124 bytes
MD5 Hash	0e954f43fd73f56e812a7285f32e41d3

Key-words associated with program/file	**`ADMsniff`**
	`libpcap`
	`Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V`
	`ADM`
	`mel`
	`^pretty^`
	`bpf_filter.c`
	`pcap-linux.c`
	`savefile.c`
	`..ooOO The ADM Crew OOoo..`
	`The_l0gz`

Screenshot showing MD5 hash values:

Screen Dump showing md5 hash, strings, and times

Program Description

Type of program

This program is a network sniffer that runs on Linux systems. This is obvious from examining the output of **`file`** and **`strings`** on the file. Running the program under the **`gdb`** debugger on an expendable system (VMWare is your friend!) reveals the following usage strings:

```
ADMsniff priv 1.0 <device> [HEADERSIZE] [DEBUG]
ex   : admsniff le0
 ..ooOO The ADM Crew OOoo..
```

Program use

Network sniffers are used to capture data from a broadcast network (this sniffer does not include mechanisms such as ARP-cache poisoning to capture data from switched networks). An attacker could use this data for password stealing and general confidentiality attacks.

Step-by-step actions of the program

At a high level, this program is fairly simple. It installs itself in the IP stack, looks at packets flowing to or from various TCP ports (listed the table below) and dumps the data into a file called **`The_l0gz`**:

TCP Ports Examined/Logged by the Program

Port	Name	Use	Authentication Mechanisms
21	FTP	File Transfers	Cleartext logname/password
23	Telnet	Remote login	Cleartext logname/password
109	POP	Remote e-mail reading	Cleartext logname/password
110	POP3	Remote e-mail reading	Cleartext logname/password
143	IMAP	Remote e-mail reading	Cleartext logname/password
512	exec	Remote command execution	Cleartext logname/password
513	login	Remote login	Cleartext logname/password or cleartext machine-name/user-name
514	shell	Remote command execution	Cleartext machine-name/user-name
1521	SQLnet	Oracle SQLnet database connectivity	Cleartext logname/password

Notice that all these ports are useful for collecting weak credentials and replaying them to break into people's e-mail/login/database accounts.

The implementation of this program is also fairly straightforward. It calls **`pcap_open_live`**() from the **`libpcap`** network packet capturing library[1] to interpose itself into the IP stack as a sniffer, calls **`pcap_next`**() from **`libpcap`** to actually sniff packets, throws out any traffic that's not associated with the TCP ports listed above, and logs whatever passes the test to **`the_l0gz`**.

`pcap_open_live`(), in turn, calls **`socket(PF_INET, SOCK_PACKET, htons(ETH_P_ALL));`** to create an endpoint socket that can grab raw packets off the network device in promiscuous mode, **`bind`**()'s the socket, and calls **`ioctl`**() to tweak the socket bindings. Note that I confirmed the system calls seen in the source-code in the output of **`strace`** on the Unknown Binary.

`pcap_next`() essentially calls **`recvfrom`**() to get packets from the socket setup by **`pcap_open_live`**(), though it can also call **`bpf_filter`**() (bundled in **`libpcap`** but originally from the Stanford/CMU **`enet`** packet filter according to the copyright notices) to do packet filtering.

The filtering code that is actually implemented grabs a packet, checks that it's a TCP packet, and then loops over a list of ports to search for (crudely implemented as an array with the value 31337 as a terminator). If the packet's source or destination port matches one of the ports in the list, a flag is set. At the end of the loop, the program dumps the packet’s contents to **`The_l0gz`** if the flag is set.

Note that this filtering code totally unnecessary given the call in **`pcap_next`**() to **`bpf_filter`**(), which can actually do much more advanced filtering[2]. However, there are reports that **`bpf_filter`**() does not work uniformly across all platforms[2], so the authors may have decided to roll their own after running into problems.

However, the code they rolled is pretty ungly… it keeps the program from being able to examine traffic on port 31337, and it doesn't scale to looking at many ports as the time signature is **`O(num-packets * num-ports)`**.

A much faster filter would be one that keeps an array of flags for all 64k ports (requires a mere 8KB memory if done with 1 bit/flag), sets the flags for all the ports its interested in, and then checks each packet against whether the port it's send from/to has its flag set. This is **`O(num-packets + num-ports)`**, which is likely to be a whole lot faster, even with small values of **`num-ports`**.

Time last used

Can't tell from the ZIP archive provided

Forensic Details

Forensic footprints

Not many footprints on installation. Basically just the creation of the file itself, and the associated M-time change on the directory it's placed in.

What other files are used when the program is executed/implemented?

Running **`strings | grep ^/`** on the file reveals that it references:

```
/lib
/usr/lib
/usr/lib/gconf
/usr/share/locale
/locale.aliases
/proc
/usr/share/zoneinfo
/dev/null
/etc/ld.so.cache
/etc/suid-debug
/proc/self/exe
/proc/sys/kernel/osrelease
/usr/lib/gconv/gconv-modules.cache
/usr/share/locale
/proc/self/cwd
/etc/mtab
/etc/fstab
/cpuinfo
/meminfo
/usr/lib/locale
/etc/localtime
```

Running under **`strace`**, however, shows that the only *actual* filesystem access is that it creates/writes a log file called **`The_l0gz`**.

How is the filesystem affected by the execution of the program?

`The_l0gz` created. Other files found by **`strings`** may be accessed.

Does the program use, manipulate, or reference any other system files?

Not evident through **`strings`** or **`strace`**. Interestingly, **`/etc/ld.so.cache`** appears in the **`strings`**, but **`ldd`** claims the file is not dynamically linked.

Are there any "leads" that could be pulled out of the file for further investigation (e.g., IP address, user information, etc.)?

No IP addresses or e-mail addresses appear in the **strings** output. However, the following bits do appear:

```
C/o Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V
+45 3122-6543
+45 3325-6543
credits: ADM, mel, ^pretty^ for the mail she sent me
 ..oOOO The ADM Crew OOoo..
```

The first line appears to be a name and address in Copenhagen, Denmark[3]. However, this name/address seems to be associated with the Linux internationalization libraries, and not with the direct creators of the binary in question. This is evident because this line appears in the various files under **/usr/lib/locale** on my Red Hat Linux version 7.1 system and these files indicate that this is the name/address of the maintainer of the Linux internationalization libraries. Further, searching for this line on Google gets you links to an awful lot of Linux binaries[4].

According the text of files under **/usr/lib/locale**, the second and third lines are apparently Mr. Simonsen's telephone numbers.

The fourth line looks like list of people involved in writing the program.

The fifth and final line confirms that these people are part of The ADM Crew.

Program Identification

I located what appears to be the "official" program source code to the **ADMsniff-1 priv 1.0** program at adm.freelsd.net/ADM/ by searching Google for **ADMsniff**[4].

There is some question whether this is the *exact* source code used for compiling this program because:

- The output of the **strings** command on the version compiled from the source found on the web is not an exact match with the Unknown Binary.
- The system call patterns of the two binaries are not identical according to **strace**, as the binary compiled from the source found on the web appears to have an extra call to **getpid()** when running against the loopback interface **lo**.

On the other hand:

- Usage messages are identical.
- Both programs claim to be version '**priv 1.0**'.
- Both programs are sniffers that capture selective traffic to **The_l0gz** (e.g., both capture Telnet traffic, neither captures HTTP traffic).
- Program sizes are reasonably similar after **strip**ping the binary I compiled (382,144 bytes for the Unknown Binary vs. 399,124 for the binary I compiled).

- The difference in system call patterns is minute and can easily be explained by compiling with different library versions.
- The source code found at adm.freelsd.net/ADM/ is an exact match with the source code to the **`ADMsniff-1 priv 1.0`** program on other ADM-related sites I found on the web such as packetstormsecurity.nl/groups/ADM/ADMsniff.tgz and www.phreak.org/archives/exploits/unix/network-sniffers/ADMsniff.tgz. And, it is a much closer match to the Unknown Binary than anything else I found on the web, including other versions of ADMsniff found on the web, such as ADMsniff 0.8 found at www.securityfocus.com/data/tools/ADMsniff-v0.8.tgz and www.cotse.com/sw/sniffers/ADMsniff-v0.8.tgz.

I believe these are the same program, but the unknown binary was built with a different compiler and/or library files. For the uninitiated, the compiler is the program used to build the actual executable machine-instruction program from the human-readable source-code, while the library files contain common executable machine-instruction routines that get linked into every program by the compiler so that the programmer doesn't have to supply the source-code to these operations to the compiler every time.

In other words, the unknown binary was built from the source-code I found on the web, but deviates from the binary I built myself from this source-code because the two compilers used generated slightly different (but functionally equivalent) executable machine instructions and/or because the compilers linked in slightly different (but functionally equivalent) executable machine-instruction library routines.

One particularly strong piece of evidence supporting this view is that the output of the **`strings`** program for the two binaries reveals differences that look exactly like differences in library versions (after filtering for lines shorter than six characters):

diff in strings output, screen 1

diff in strings output, screen 2

Legal Implications

Proof Program Executed

It is possible to prove that the program was executed based on the existence of the **`The_10gz`** file (assuming proper incident/forensic precautions were taken so that it could be shown that the evidence gathered had a specific chain of custody, etc.). Therefore, a discussion of legal implications is appropriate here.

Of course, one could argue that someone placed an empty **`The_10gz`** file on the system when installed the sniffer software. But, that argument would be hard to make if **`The_10gz`** contained data obviously taken from the local network.

Applicable Laws

My particular circumstances are such that the compromised machine would be located in the United States, and would likely contain medical information (I work for a nationwide healthcare company).

So, assuming that the program were placed by an attacker and not by an agent of the company (e.g., a system administrator or some other person with "permission" to place a sniffer), then the attacker would be in violation of

- The Computer Fraud and Abuse Act (CFAA)[5]
- The Wiretap Act (WA)[6].
- and The Electronic Communications Privacy Act (ECPA)[7].

Depending on the nature of the data found in the **`The_10gz`** file, one could also argue that the attacker might be in violation of

- The Criminal Trade Secrets statue 18 U.S.C. 1832[8].

Moreover, depending on who carried out the attack and what their motives where, they might also be in violation of

- The Economic Espionage Act 18 U.S.C. 1831[9]

Since we have the medical records of U.S. Military personnel on our systems (the DoD is a client), an interesting question is whether the records could be used in such a way that the perpetrator would be in violation of Treason, Sedation, and Subversion statutes under 18 U.S.C. Chapter 115[10]. Consensus among my colleagues is that is possible to come up with Treason scenarios, such as using the knowledge that a certain battlefield commander is HIV positive to blackmail them into changing their actions. Statutes covered under such situations might be:

- 18 U.S.C. 2381 – Treason[11],
- 18 U.S.C. 2383 - Rebellion or insurrection[12],
- 18 U.S.C. 2384 - Seditious conspiracy[13],
- 18 U.S.C. 23987 - Activities affecting armed forces generally[14],
- and 18 U.S.C. 2388 - Activities affecting armed forces during time of war[15].

Another interesting issue to consider is how difficult it would be to meet the "Protected Computer" and "damage" requirements under CFAA[5]. It would not be difficult to argue that our computers are used-in/affect "*interstate or foreign commerce*" and that attacking them would necessarily cost us at least $5,000 (if only in the cost of the investigation). More interestingly, however, we can point out the existence of medical records and argue that their disclosure could "*potentially modif[y] or impair ... the medical examination, diagnosis, treatment, or care of one or more individuals*," [5] eliminating the need to show $5,000 in damages or the involvement in interstate/foreign commerce under the CFAA.

On the state side, a quick search on the web site for the Legislature of the State of New Jersey (where my office and most of our computer infrastructure are located) shows that the attacker might also be in violation of

- Computer related theft 2C:2-25, 2C:2-25, and 2C2-27[16].

Note that I cited three statutes here because 2C:20-25 is the base statute, and -26 and -27 describe additional penalties for various degrees of theft.

Other states my company operates computer systems in include Florida, Pennsylvania, Texas, and Nevada. Those states likely have similar statutes.

Nevertheless, the Federal statutes are likely to override any use of state statues to prosecute a case such as this given the likelihood of meeting the CCFA requirements as discussed above.

Penalties

Similar cases have garnered punishment anywhere from slap-on-the-wrist community service, to serving several years in Federal prison. The following table elaborates on punishments under various Federal Statutes[5] [6] [7] [8] [9] [11] [12] [13] [14] [15]. I would have also added the NJ statutes, but they do not actually specify penalties.

Statute	Minimum Penalty	Maximum Penalty
Computer Fraud and Abuse Act	fine	20 years imprisonment
Wiretap Act	$500 fine per violation	five years imprisonment
Electronic Communications Privacy Act	fine	two years imprisonment
Commercial Trade Secrets	fined	ten years imprisonment (individuals) $5,000,000 (organizations)
Economic Espionage		$500,000 fine and/or 15 years imprisonment (individuals) $10,000,000 fine (organizations)
Treason	five years imprisonment and fined $10,000, and *incapable of holding any office under the United States*	
Rebellion or insurrection	fined and *incapable of holding any office under the United States*	ten years imprisonment

Seditious conspiracy	fined	20 years imprisonment
Activities affecting armed forces generally	fined, *ineligible for employment by the United States or any department or agency thereof, for the five years next following his* [sic] *conviction*	five years imprisonment
Activities affecting armed forces during time of war	fined	20 years imprisonment

Authorized Use

The discussion so far is based on the possibility that person who activated the sniffer was an attacker and not an agent of the company. So, the question is, could they have broken any laws if they were an agent of the company? In my opinion, the answer is Yes. The System Administrator and/or anyone whom authorized/directed them to put the sniffer in place may be in violation of Wiretap Act. See **Part 3 - Legal Issues of Incident Handling** for a complete treatment of why. As for penalties here, this would likely be limited to civil penalties to recover damages.

Another interesting question is whether anyone else might be criminally liable. Under the proposed security requirements of the Health Insurance Portability and Accountability Act (HIPAA) agents of the company may be liable when medical data is made public if they have been negligent in protecting it[17]. This could come into play in a situation where an attacker was able to pick up medical data using a sniffer.

Internal Policies

On the Policy side, we are a bit of a mess owing to our imminent spin-off from a large company with very mature policies into our own entity. So it's possible that no formal policies would have been violated.

One area where there are strong policies is in the Policy requiring the use of Computer Systems for Business Use. However, this is directly contradicted by our Ethics policy and our Internet usage policy, which explicitly allow the use of company resources for personal use as long as "*such use has no adverse effect on productivity and the work environment*."

Another possibility is the Internet and Other Communication Tools Policy, but it only takes effect if information obtained by the sniffer is disclosed to the outside world through electronic means.

Interview Questions

This probably depends greatly on whether I believe interviewee is a well-intentioned insider, a malicious insider, or a malicious outsider. At first I was going to say that I'd do questions for the well-intentioned insider. But, it's really not that useful for well-intentioned usage given that the software filters much of the traffic, On the other hand, we don't have to let the interview subject know that...

- Did you hear that someone installed a sniffer program on XYZ? The way management keeps expecting us to get things done faster and faster, but keeps putting red tape in our way, I bet they were just trying to debug a networking problem and couldn't get a hold of the Networking Services team.
- Still, there's some very sensitive data on that network, and we don't know for sure what their motives were. You don't happen to know anything about it, do you?
- I sure hope they come forward so we can be sure there are no other sniffers lurking around. Getting this behind us is the most important thing right now. Still, they really should have gone through Networking Services. You know, we're going to have to turn the case over to the FBI if we don't find out what happened pretty soon. Did you know they could be violating the US Wiretap Act? That's pretty serious.
- Wow, I'm glad you told me that you were the one who put that in place. Now we can go clean it up and put it behind us. Oh yeah, did you install sniffers on any other systems? We need to make sure they're all cleaned up so we can be sure nobody's misusing the data.
- By the way, it looked like you setup the sniffer to get just the info you needed. That's pretty clever. How did you do that?

Additional Information

The single biggest resource I used was Google (www.google.com).

From there, I accessed the various sites listed in the Program Identification section looking for the source code to the Unknown Binary.

I also accessed the Cornell University Legal Information Institute's collection of the U.S. Code on the web (www.law.cornell.edu/uscode), the legal sections of the U.S. Internet Industry Association's web site (www.usiia.org/legis/legis.htm), the web site of Computer Professionals for Social Responsibility (www.cpsr.org), the Jones International Telecommunications and Multimedia Encyclopedia (www.jonesencyclo.com/encyclo/), The New Jersey Legislature (www.njleg.state.nj.us/), and HIPAAdvisory (www.hipaadvisory.com) for legal information.

Finally, I used the IANA Port Numbers list (www.iana.org/assignments/port-numbers) the SANS Intrusion Detection FAQ (www.sans.org/newlook/resources/IDFAQ/oddports.htm), and the G-Lock Software Trojan Ports list (www.glocksoft.com/trojan_port.htm) to find port number assignments.

References

1. The Tcpdump Group, manual page for "pcap – Packet capture library." Available from www.tcpdump.org/pcap3_man.html.
2. Tim Carstens, "Programming with Pcap." Available from www.tcpdump.org/pcap.htm.
3. Københavns Kommunes kampagne-website, www.kobenhavn.dk
4. Various searches on Google in June and July 2002, www.google.com.
5. The Computer Fraud and Abuse Act, 18 U.S.C. 1030, "Fraud and related activity in connection with computers." Available from www4.law.cornell.edu/uscode/18/1030.html.
6. The Wiretap Act, 18 U.S.C. 2511, "Interception and disclosure of wire, oral, or electronic communication prohibited." Available from www4.law.cornell.edu/uscode/18/2511.html.
7. The Electronic Communications Privacy Act, 18 U.S.C. 2701, "Unlawful access to stored communications.". Available from www4.law.cornell.edu/uscode/18/2701.html.
8. The Criminal Trade Secrets Act, 18 U.S.C. 1832, "Theft of trade secrets." Available from www4.law.cornell.edu/uscode/18/1832.html.
9. The Economic Espionage Act, 18 U.S.C. 1831, "Economic espionage." Available from www4.law.cornell.edu/uscode/18/1831.html.
10. 18 U.S.C. Chapter 115, "Treason, Sedition, and Subservive Activities." Available from www4.law.cornell.edu/uscode/18/pIch115.html.
11. 18 U.S.C. 2381 – "Treason." Available from www4.law.cornell.edu/uscode/18/2381.html.
12. 18 U.S.C. 2383 – "Rebellion or insurrection." Available from www4.law.cornell.edu/uscode/18/2383.html.
13. 18 U.S.C. 2384 – "Seditious conspiracy." Available from www4.law.cornell.edu/uscode/18/2384.html.
14. 18 U.S.C. 23987 – "Activities affecting armed forces generally." Available from www4.law.cornell.edu/uscode/18/2387.html.
15. 18 U.S.C. 2388 – "Activities affecting armed forces during time of war." Available from www4.law.cornell.edu/uscode/18/2388.html.
16. NJ Permanent Statutes 2C:20-25, 2C:20-26, and 2C:20-27 – "Computer related theft" available from www.njleg.state.nj.us/.
17. Federal Register of the United States, "Health Insurance Portability & Accountability Act of 1996 (HIPAA)." Available from aspe.hhs.gov/admnsimp/Index.htm. Additional information on security provisions available from aspe.hhs.gov/admnsimp/Index.htm, www.hcfa.gov/hipaa/hipaahm.htm, and www.hipaadvisory.com/regs/privacynprm/index.htm.

Part 3 - Legal Issues of Incident Handling

Wiretap Statute - Wiretap Act exemptions for system administrators

Related Statutes and their Exemptions

Wiretap Act – U.S.C. 2511 - Interception and disclosure of wire, oral, or electronic communications prohibited[1]

This statute is what is generally referred to as The Wiretap Act (though it actually replaces an earlier statute, 18 U.S.C. 1334, that went by the same heading). The primary exemption to the Wiretap Act that a system administrator might work under would be[1]

> *"It shall not be unlawful under this chapter for an operator of a switchboard, or an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service, except that a provider of wire communication service to the public shall not utilize service observing or random monitoring except for mechanical or service quality control checks."*

Under this exemption, I would have broad authority to carry out network sniffing and other forms of "wiretap" related to my duties in administering the systems/networks I have domain over. This would include things like investigating alleged abuses of computing/network resources by other employees (in that I would be protecting the rights/property of my employer).

Note, however, that the Act makes it clear that this doesn't allow me to violate other laws in the following of the Act[1]:

> *"It shall not be unlawful under this chapter for a person not acting under color of law to intercept a wire, oral, or electronic communication where such person is a party to the communication or where one of the parties to the communication has given prior consent to such interception unless such communication is intercepted for the purpose of committing any criminal or tortious act in violation of the Constitution or laws of the United States or of any State."*

USA Patriot Act - Public Law 107-56, Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001[2]

Among many other things, the USA Patriot Act amends the Wiretap Act to allow System Administrators to call upon law enforcement to help them monitor their networks when they suspect a crime is being committed. This is in contrast with the previous reading of the Wiretap Act (cited above) which exempted those acting "*under the color of law*" [1] in the exemptions.

Note, however, that there are organizations such as the Electronic Frontier Foundation[3] and the ACLU[4] that have called into question the constitutionality of some aspects of the USA Patriot Act, though not this particular provision. Therefore, it would be wise for system administrators to get sign-off from both their management and their corporate council before calling in law enforcement for monitoring help.

Electronic Communications Privacy Act (ECPA) – 18. U.S.C. 2701 - Unlawful access to stored communications[5]

As the name of this statute implies, ECPA is not about wiretapping, but rather about stored communications (such e-mail or v-mail stored on a server).

One interesting question is where the Wiretap Act ends and ECPA takes over. This is important because it governs things like whether examining data on an e-mail server is the same as intercepting an e-mail in transit. Or whether examining data on a web server is the same as intercepting web traffic in transit.

At first blush, the courts appear undecided here.

The ruling in *Eagle Investment Systems Corporation v. Einar Tamm, et al., 2001 U.S. Dist. Lexis 7349 (D. Mass., May 22, 2001)* [6] was that accessing an e-mail sitting on an e-mail server after the mail had been read (but not explicitly deleted) was not a wiretap (i.e., covered by ECPA and not the Wiretap Act).

Whereas the final ruling in *Robert Konop v. Hawaiian Airlines, Inc., No. 99-55106 (9th Cir. January 8, 2001), withdrawn (9th Cir., August 28, 2001)* was that unauthorized access to data on a password-protected web site was a wiretap[7]. Furthermore, the court specifically stated that[7]:

> *"It is perfectly clear that the framers of the Wiretap Act's current definition of "electronic communication" understood that term to include communications in transit and storage alike. ... It makes no more sense that a private message expressed in a digitized voice recording stored in a voice mailbox should be protected from interception, but the same words expressed in an e-mail stored in an electronic post office pending delivery should not. We conclude that it would be equally senseless to hold that Konop's messages to his fellow pilots would have been protected from interception had he recorded them and delivered them*

> *through a secure voice bulletin board accessible by telephone, but not when he set them down in electronic text and delivered them through a secure web server accessible by a personal computer. We hold that the Wiretap Act protects electronic communications from interception when stored to the same extent as when in transit.*"

An astute reader, however, will notice that these two rulings are not actually at odds, since Eagle refers to messages that were read but not deleted from the server, while Konop refers to messages that had not yet been read by all potential recipients, something both courts were careful to point out in their rulings.

So, it appears that the Wiretap Act covers data until it is delivered to the (human) recipient. But, ECPA covers the data once the recipient receives the data and decides to save it, as it is no longer "in transit" to the recipient.

In either event, the ECPA exemptions for System Administrators are largely similar to those in the Wiretap Act[1]:

> "*Subsection (a) of this section does not apply with respect to conduct authorized –*
> *(1) by the person or entity providing a wire or electronic communications service;*
> *(2) by a user of that service with respect to a communication of or intended for that user; or*
> *(3) in section 2703* [Requirements for Governmental Access]*, 2704* [Backup preservation] *or 2518* [Procedures for interception of wire, oral, or electronic communications] *of this title*"

Do these exemptions allow random monitoring?

One important question is whether these exemptions allow me to do random/wholesale monitoring of traffic to find potential abuses. The Wiretap Act clearly does not exempt this for a "*provider of wire communication service to the public*"[1] such as an ISP or Telco. But I don't work for an ISP or Telco.

In my particular circumstances, there are two issues: intercepting private communications of other employees, and intercepting communications of the general public accessing our web sites.

Employee communications

On the face of it, it should be clear from the text of the Wiretap Act that I can intercept any employee communications as long as I can show that it was in the interest of protecting the company's rights/property. However...

- The 6th U.S. Circuit Court of Appeals ruled in *Adams v. City of Battle Creek, 6th U.S. Circuit Court of Appeals, No. 99-1543 (2001)* that a police department can not use wiretaps that are not part of its "*ordinary course of business*" to

investigate specific allegations against an officer (in this case intercepting pager communications to investigate charges that the officer was assisting drug dealers)[8]. But then, the Wiretap Act has very different restrictions for law enforcement than for other employers.
- However, 9th U.S. Circuit Court of Appeals ruled in *Robert Konop v. Hawaiian Airlines, Inc., No. 99-55106 (9th Cir. January 8, 2001), withdrawn (9th Cir., August 28, 2001)* similarly concluded that a commercial entity could not use wiretaps outside its ordinary course of business to investigate union involvement of its employees[7].
- And, a Maryland court ruled in *Schmerling v. Injured Workers' Insurance Fund, Md. App., No. 88, (4/8/02)* that the Maryland Wiretap Act does not allow the installation of equipment (and presumably software) for the sole purpose of monitoring employees[9]. Again, one could argue that this is subordinate to the Federal Act, but the point is that the courts seem to have a reasonably high standard for employers showing that their monitoring was in the ordinary course of business and not specifically for monitoring the general activity employees.

So, it looks reasonably certain that I *could* monitor employee communications as long as the monitoring was being done to protect the rights/property of the company (such as detecting patterns of intrusion, virus scanning, detecting leakage of proprietary data, etc.). But, not *specifically* to monitor private employee communications that have no *direct* impact on the company.

However, it is possible that California courts would still require specific consent on the part of the employees. I will address the issue of what constitutes consent below.

Web site access by the general public

Our web servers accept messages destined for specific other humans (prescription orders destined to pharmacists and "feedback" destined to customer-service representatives). Therefore, the same monitoring restrictions discussed in the Employee Communications section seem to apply to these communications.

Further, one might argue that operating a public web server implies that the company is a "*provider of wire communication service to the public*"[1], in which case general random monitoring would clearly be in violation of the Wiretap Act. However, monitoring using mechanical systems such as IDS and virus scanners would still be exempted. So, the Wiretap Act here would seem to restrict what individuals may access the data not what the "mechanical systems" of the company can do with the data. Meanwhile, a similar reading of ECPA would seem to protect data stored and collected by web site users of the general public. Thus implying a limitation to the ability of companies to disseminate information obtained from the public without specific consent (such as selling mailing lists). The Wiretap Act and ECPA could also be used under these circumstances to determine penalties should the company inadvertently disclose such data through a web site breach.

So, to allow a greater level of monitoring, disclosure, mailing list sales, etc. the company should get specific consent from its web site users. In addition, it should get specific consent from any employees that specific messages from its web site are destined for (to cover state laws requiring consent of both the sending and receiving parties).

Gaining Consent

This section offers of brief taxonomy and comparison of mechanisms for obtaining such consent from employees and web-site users.

Mechanism	Description	Does it work for Employee Communications?	Does it work for a Web Site?
Written policy	Description of monitoring policies published in Employee Handbook or Web Privacy policy	Only if you can show that everyone actually read the policy	No, since most attackers do not bother to read the privacy policy…
Click-Through or Signed Paper	As above, but with a click-through or paper signature showing that the user has seen the policy	Yes, since now you can positively show the consent implied by clicking/signing	Yes, for applications that deny access without the click-through
Port Bannering	Post a mini version of the policy in a banner that is shown on every access to a network service	Maybe. Guarantees the policy is posted on every access. But no case law showing this actually implies consent, hard/impossible to banner some applications, and no opportunity to see the banners in some other applications. Not to mention that automated attack tools ignore the banners.	

Judging from the above, it seems the right thing to do is all of the above. I.e.,

1. Put monitoring information into all relevant policy
2. Get employee's signoff on monitoring policies
3. Use a click-through of the policy where practical on web sites
4. Banner wherever practical

References

1. 18 U.S.C. 2511, Interception and disclosure of wire, oral, or electronic communications prohibited . Available from www4.law.cornell.edu/uscode/18/2511.html.
2. Public Law 107-56, Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001
3. Electronic Frontier Foundation, "EFF Analysis of the Provisions of the USA Patriot Act that relate to Online Activities," 31 October 2001. Available from www.eff.org/Privacy/Surveillance/Terrorism_militias/20011031_eff_usa_patriot_analysis.html.
4. American Civil Liberties Union, "USA Patriot Act Boosts Government Powers while Cutting Back on Checks and Balances." Available from www.aclu.org/congress/l110101a.html.
5. 18 U.S.C. 2701, Unlawful access to stored communications
6. Martin H. Samson, synopsis of *Eagle Investment Systems Corporation v. Einar Tamm, et al., 2001 U.S. Dist. Lexis 7349 (D. Mass., May 22, 2001)*. Available from www.phillipsnizer.com/int-art232.htm.
7. Martin H. Samson, synopsis of *Robert Konop v. Hawaiian Airlines, Inc., No. 99-55106 (9th Cir. January 8, 2001), withdrawn (9th Cir., August 28, 2001)*. Available from www.phillipsnizer.com/int-art225.htm.
8. policecenter.com, "Department taps officer's pager." Available from www.policecenter.com/rulings/r052301api.shtml.
9. Criminal Practice, "Surveillance: Employer's Recording System Violates Wiretap Act." Available from criminalpractice.pf.com/subscribers/html/showarticle.asp?article=820.