BORIS LOZA

under attack

DEALING WITH MISSING UNIX FILES

Boris Loza is a founder of Tego System Inc. and HackerProof Technology, in addition to being a contributor to many industry magazines. He holds several patents and is an expert in computer security. He loves nature, reading books, and watching movies, and enjoys scuba diving and entomology.

■ *bloza@hackerproofonline.com*

A SECURITY BREACH CAN INSPIRE panic in administrators. This quick application note explains some techniques to be used to recover the names and contents of files during an attack or shortly thereafter.

Takedown, by Tsutomu Shimomura and John Markoff, describes the pursuit of Kevin Mitnick by Tsutomu Shimomura. It notes: "I could make out patterns of information still stored on my computer's disk that revealed the ghost of a file that had been created and then erased. Finding it was a little like examining a piece of paper on a yellow legal pad: even though the top page has been torn off, the impression of what was written on the missing sheet can be discerned on the remaining page."

In the UNIX and Linux worlds, just about everything is a file. UNIX treats regular files, directories, hard disks, printers, modems, and so on as files. When a file is created, it is assigned an inode (an index structure that is quicker for finding on-disk data structures than filename matching). When a file is deleted, the inode number is cleared from the directory, but the file does not vanish. The contents usually remain on the disk, at least for a while, until the disk blocks containing the contents are reused.

Listing Deleted Files

Because a directory is also a file, commands that manipulate files can be used to examine a directory. The od command performs an octal dump, which can include an ASCII listing. Let's consider an example of a directory (all outputs are taken from Solaris 9, except where otherwise indicated):

```
$ chdir testdir
$ ls -a
.       ..       Project    status     webstat.log
```

The ls command says the directory contains five entities: Project, status, webstat.log, an entry for the directory itself (.), and an entry for the parent directory (..). The on-disk structure can be examined using the od command (with a flag to display the ASCII characters, if the octal code is reasonable). Certain non-graphic characters appear as the intuitive C-language escapes; other non-printable characters appear as 3-digit octal numbers.

```
$ od -c .
0000000  \0  \b 023 337  \0  \f  \0 001    . \0  \0  \0   \0 013 006   P
0000020  \0  \f  \0 002   .  . \0  \0   \0  \b 023 340   \0 020  \0 007
0000040   P   r   o   j   e   c   t \0   \0  \b 023 341   \0 024  \0 013
0000060   w   e   b   s   t   a   t   .   l   o   g \0   \0  \b 023 342
0000100 001 304  \0 006   s   t   a   t   u   s \0  \0   \0  \0  \0  \0
0000120  \0  \0  \0  \0  \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0
*
0001000
$
```

The output starts each line with the number of bytes, expressed in octal, shown since the start of the file. The first line starts at byte 0. The second line starts at byte 20 (that's byte 16 in decimal, the way most of us count), and so on. One can easily see the listed file names on this output. It is easy to see that each file name is preceded by 8 bytes of some sort of file-system information. Additionally, file names are padded to the next 32-bit boundary (maybe after a \0 that terminates the file name).

Let's delete the status file and compare the od -c output with the previous one.

```
$ rm status
$ ls -a
.          ..          Project      webstat.log
$ od -c .
0000000  \0  \b 023 337   \0  \f  \0 001    . \0  \0  \0   \0 013 006   P
0000020  \0  \f  \0 002    .  . \0  \0   \0  \b 023 340   \0 020  \0 007
0000040   Project  \0   \0  \b 023 341  001 330  \0 013
0000060   webstat.log  \0   \0  \0  \0  \0
0000100 001 304  \0 006    status  \0  \0   \0  \0  \0  \0
0000120  \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0   \0  \0  \0  \0
*
0001000
```

Note that we still can see the status file, but four of the eight bytes that precede it (\b, \023, and \342) have been replaced by NULs, a zeroing of the inode number that links the name to the on-disk storage.

Understanding the Output

To understand more about all these outputs, let's take a look at the UNIX File System (UFS) directory structure that can be found in /usr/include/sys/ufs_fsdir.h:

```
struct  direct {
   uint32_t   d_ino;                /* inode number of entry */
   u_short    d_reclen;              /* length of this record */
   u_short    d_namlen;              /* length of string in d_name */
   char       d_name[MAXNAMELEN + 1]; /* name must be no longer than
this */
};
```

Now, we may better understand the output from od -c. We will analyze the initial output that was displayed above:

```
$ od -c .
0000000  \0  \b 023 337  \0  \f  \0 001   . \0  \0  \0  \0 013 006   P
0000020  \0  \f  \0 002   .  . \0  \0  \0  \b 023 340  \0 020  \0 007
0000040   Project  \0  \0  \b 023 341  \0 024  \0 013
0000060   webstat.log  \0  \0  \b 023 342
0000100 001 304  \0 006   status  \0  \0  \0  \0  \0  \0
0000120  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0001000
```

As you can see from this output, and based on the UFS directory structure, one can recognize the file names: d_name[MAXNAMELEN +1] (., .., Project, webstatus.log, and status). Before each file name, we can see a number that represents the length of string in d_name—001 for ., 002 for .., 007 for Project, 013 (11 decimal) for webstat.log, and 006 for status.

What can also be recognized in this output are two octal digits that represent the inode number in the list of inodes, d_ino. This is 013 006 for .., \b 023 337 for ., \b 023 340 for Project, and so on.

At this point, programmers might well think about writing a quick program to traverse the directory structure to print the files, maybe even flagging those with inode numbers of zero. Regrettably, the indexing information on some OSes is adjusted when a file is deleted. A better approach is simply to text-process the strings in the directory.

Probably the easiest way to do this is the strings command:

```
$ strings . > /tmp/a
$ cat /tmp/a
.
..
Project
webstat.log
status
```

It is quite possible to learn the names of deleted files (presuming the directory slots haven't already been reused). What about the data contained in deleting files?

Recovering Certain Text Files

In most UNIX and UNIX-like file systems, files are not necessarily recoverable after deletion. Sometimes, though, one can retrieve vital information.

LOG FILES

If you suspect that an intruder has modified or deleted your log file, you might try to recover what has been deleted from this file. When a file is deleted under the UNIX system, the inode number in the directory is set to 0, and disk blocks belonging to a file are marked "free" and returned to a pool of blocks that may be reused by the system. If the blocks of the deleted or altered file were not yet reused, this information is still present on the hard disk—somewhere.

Let's try to recover a log file. Assume that we log all network connections to /var/adm/messages with inetd -t. We suspect that an intruder modified our /var/adm/messages file to hide successful connections on June 4. Current output from /var/adm/messages:

```
$ tail messages
Jun  3 10:49:42 birch inetd[132]: exec[25727] from 10.56.49.194 2199
Jun  3 11:29:25 birch inetd[132]: exec[28958] from 10.56.49.194 2254
Jun  3 11:35:14 birch inetd[132]: telnet[29398] from 10.56.49.194 2255
Jun  3 14:04:21 birch inetd[132]: telnet[9711] from 10.56.53.55 57779
Jun  3 14:21:30 birch inetd[132]: ftp[10914] from 10.56.49.194 2430
Jun  3 14:51:04 birch inetd[132]: telnet[17225] from 10.56.49.194 2486
Jun  3 14:56:35 birch inetd[132]: telnet[17622] from 10.56.49.194 2487
Jun  5 09:55:00 birch inetd[132]: exec[14029] from 10.56.49.194 3248
Jun  5 11:13:33 birch inetd[132]: exec[19439] from 10.56.49.194 3281
Jun  6 14:17:14 birch inetd[132]: telnet[10520] from 10.56.49.194 3747
```

We can see that entries from Jun 4 are missing, a suspicious circumstance. Because, as we already know, almost everything in UNIX is a file, we can use the grep utility against the raw disk device on which /var/adm/messages is located for the string "Jun 4".

First, we must deduce which disk device holds (or held) the /var/adm/messages file (this output is produced on Solaris 2.6):

```
$ df /var/adm/messages
Filesystem        Kbytes       used       avail  capacity Mounted on
/dev/dsk/c0t0d0s4  290065       93826  167233 36%    /var
```

We will use grep against the entire /dev/dsk/c0t0d0s4 partition, all 2.9GB of it:

```
$ su -
Password:
Sun Microsystems Inc.  SunOS 5.6   Generic August 1997
# grep 'Jun  4' /dev/dsk/c0t0d0s4 > /tmp/123
# head /tmp/123
Jun  4 09:16:54 sundvl25 inetd[132]: telnet[2872] from 10.32.112.159
    1137
Jun  4 09:23:06 sundvl25 inetd[132]: telnet[3306] from 10.32.112.159
    1140
Jun  4 10:07:44 sundvl25 inetd[132]: ftp[6484] from 10.56.49.183 1072
Jun  4 10:08:17 sundvl25 inetd[132]: ftp[6519] from 10.56.49.183 1073
Jun  4 09:16:54 sundvl25 inetd[132]: telnet[2872] from 10.32.112.159
    1137
Jun  4 09:23:06 sundvl25 inetd[132]: telnet[3306] from 10.32.112.159
    1140
Jun  4 10:07:44 sundvl25 inetd[132]: ftp[6484] from 10.56.49.183 1072
Jun  4 10:08:17 sundvl25 inetd[132]: ftp[6519] from 10.56.49.183 1073
```

We redirected the output from the grep to the /tmp/123 file (choose any meaningless name in case the intruder is on the system), which is stored on a different partition of the hard disk. This will ensure that the data we are trying to recover is not overwritten and will avoid an embarrassing grep feedback loop.

Now, while the blocks that contain the file's contents might not be scanned in the proper order (there are no relevant rules about order of disk block allocation in a file system that has been in production for a while), we can see many of the entries that had been deleted from the /var/adm/messages file (based on the format of the messages file) by our intruder.

This process is time-consuming. Be patient. On a busy system it could take a lot of time to grep through the disk's blocks.

RECOVERING BINARY FILES

You may easily recover an executable file if it is still running as a process on your system. There may be situations in which a hacker runs the application and deletes an executable file. While the file's name is removed from a directory, the contents are still intact so that execution can proceed.

On Solaris and others, a link to the process exists in the /proc/[PID]/object/a.out directory. You may identify the process number for the deleted file by using the ps command or lsof utility.

For example, let's assume that we are going to restore a file that belongs to the process ID 22889 from the suspicious srg application that we found running on our system:

```
# ps -ef | more
UID    PID  PPID  C    STIME TTY     TIME CMD
root     0     0  0   May 10 ?       0:00 sched
root     1     0  0   May 10 ?       2:21 /etc/init -
...
root 22889 16318  0 10:09:25 pts/1    0:00 ./srg
root 16318 25824  0 08:56:27 pts/1    0:00 csh
```

As long as the command is still running, use cp to recreate its executable:

```
# cp /proc/2289/object/a.out /tmp/srg
```

The /proc directory connects to a pseudo-filesystem that abstracts the kernel's process architecture; /proc/2289/object/a.out is the process's executable binary.

You may also want take a look at Dan Farmer and Wietse Venema's The Coroner's Toolkit (TCT), which includes unrm and lazarus applications to automate this process.

Conclusion

Do not be frustrated if you cannot recover a deleted or altered file. Sometimes the odds of recovering a file are very slim. Be patient and take everything with a sense of humor!