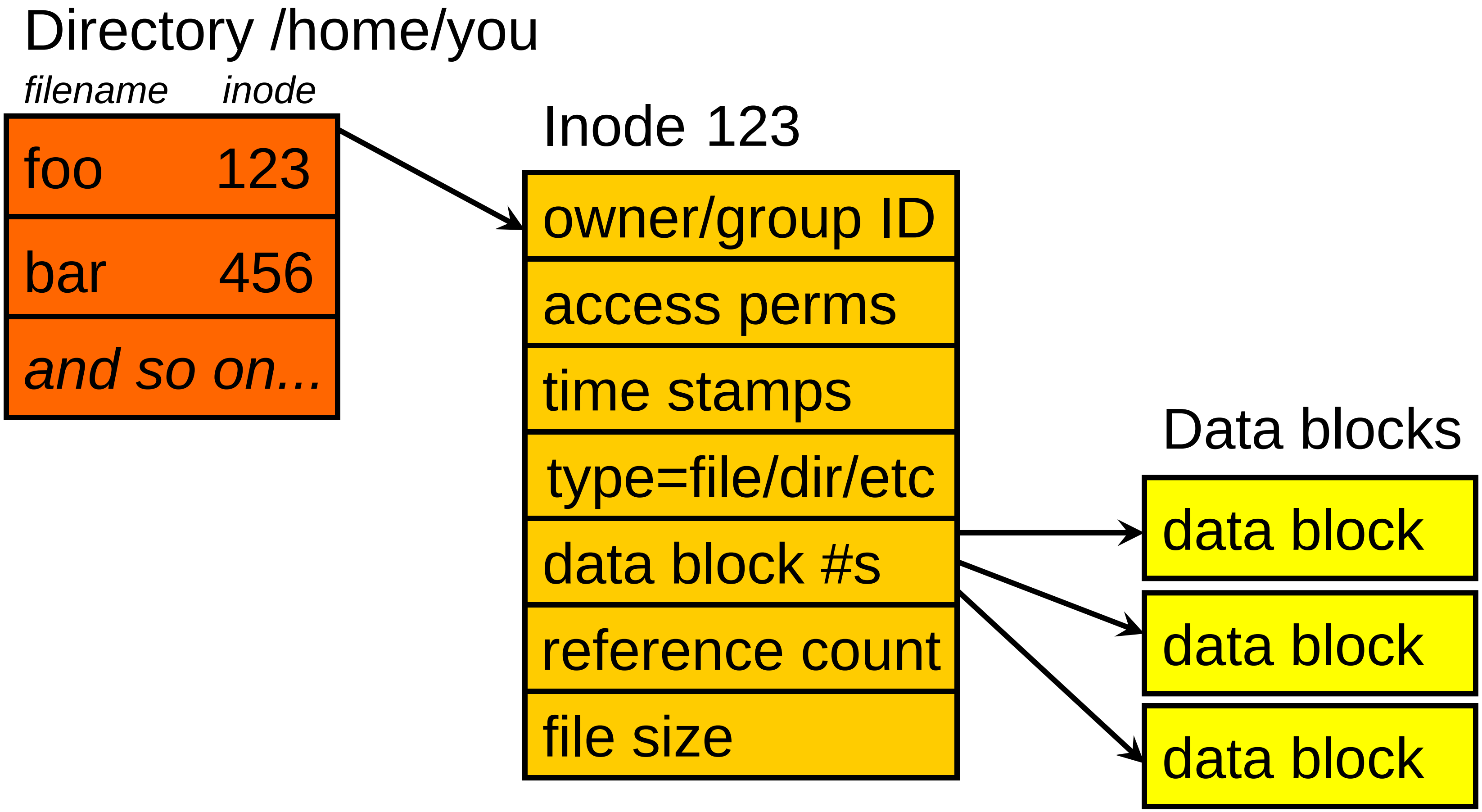
The broken file shredder
Programming traps and pitfalls

Wietse Venema
IBM T.J.Watson Research Center
Hawthorne, NY, USA

Overview

- What happens when a (UNIX) file is deleted.
- Magnetic disks remember overwritten data.
- How the file shredding program works.
- How the file shredding program failed to work.
- “Fixing” the file shredding program.
- Limitations of file shredding software.

UNIX file system architecture



Directory /home/you

filename *inode*

foo	123
bar	456
and so on...	

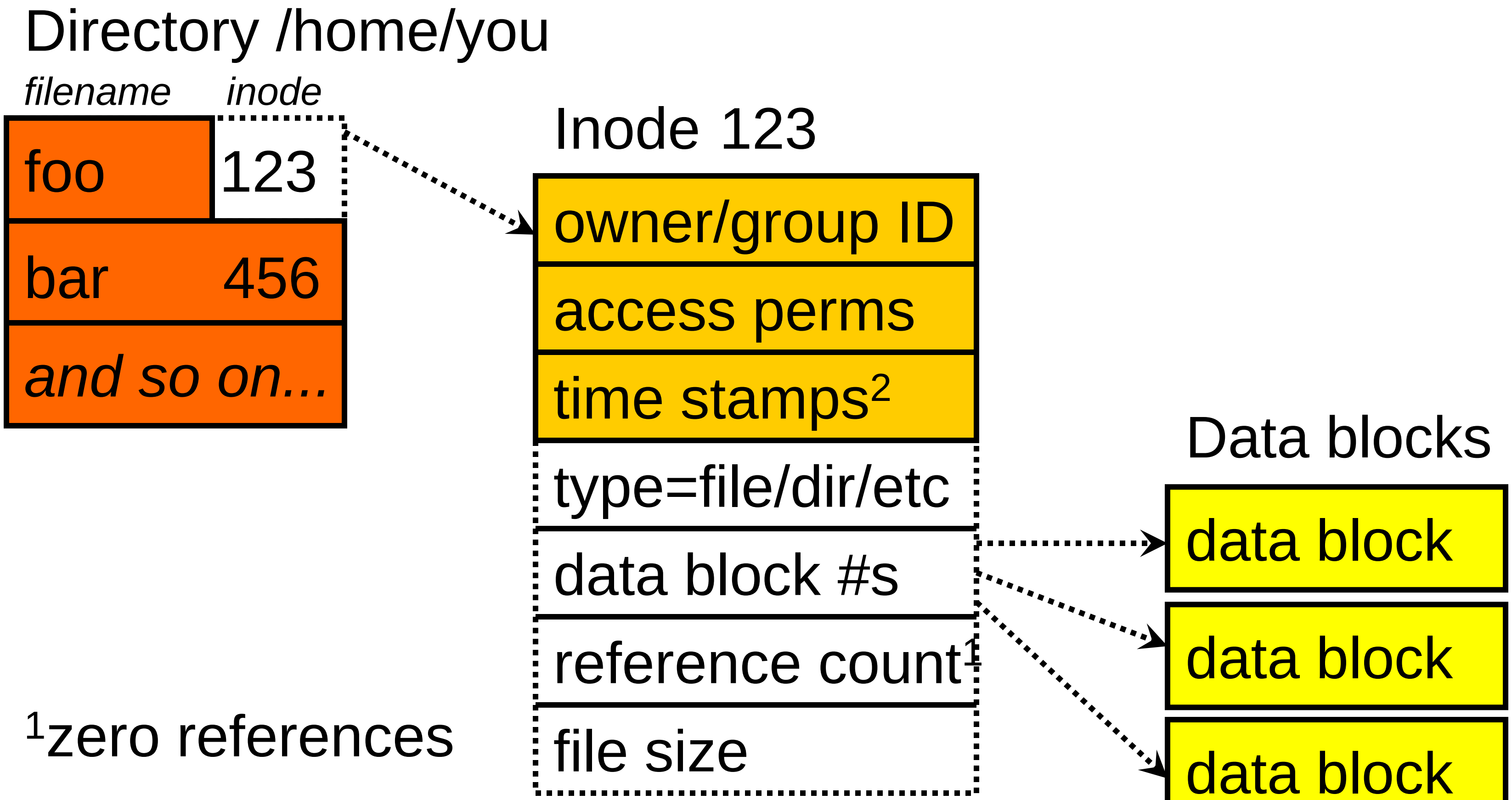
Inode 123

- owner/group ID
- access perms
- time stamps
- type=file/dir/etc
- data block #s
- reference count
- file size

Data blocks

- data block
- data block
- data block

Deleting a UNIX file destroys structure, not content



Directory /home/you

filename *inode*

foo 123

bar 456

and so on...

Inode 123

owner/group ID

access perms

time stamps[2]

type=file/dir/etc

data block #s

reference count[1]

file size

Data blocks

data block

data block

data block

[1] zero references

[2] status change time = time of deletion

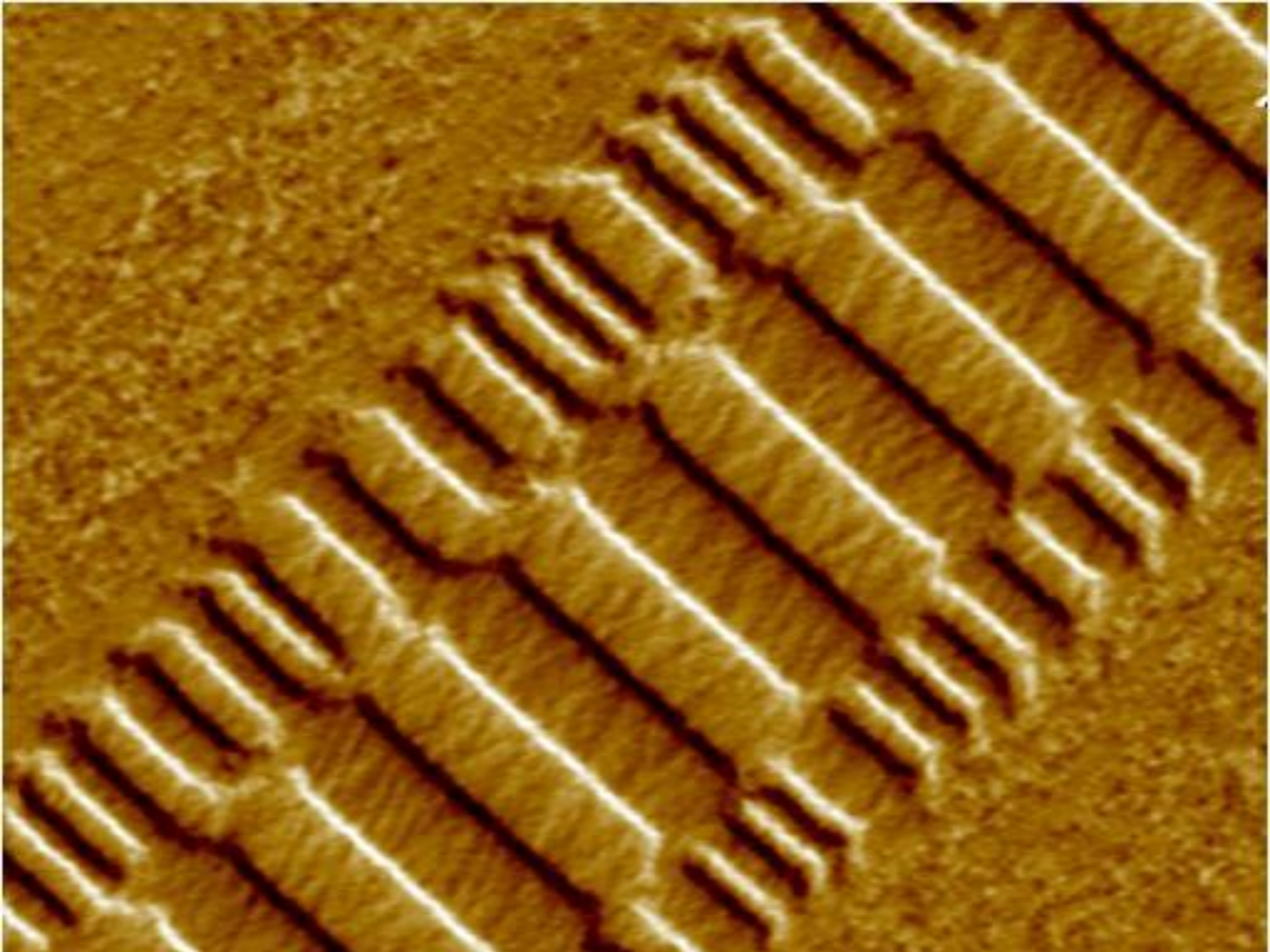
Persistence of deleted data

- Deleted file attributes and content persist in unallocated disk blocks.
- Overwritten data persists as tiny modulations on newer data.
- Information is digital, but storage is analog.

Peter Gutmann’s papers: http://www.cryptoapps.com/~peter/usenix01.pdf

and http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html

kool magnetic surface scan pix at http://www.veeco.com/ nanotheather



Avoiding data recovery with magnetic media

- Erase sensitive data before deleting it.
- To erase data, repeatedly reverse the direction of magnetization. Simplistically, write *1*, then *0*, etc.
- Data on magnetic disks is encoded to get higher capacity and reliability (MFM, RLL, PRML, ...). Optimal overwrite patterns depend on encoding.

mfm = modified frequency modulation; rll = run length limited;

prml = partial response maximum likelihood

File shredder pseudo code

```
/* Generic overwriting patterns. */
patterns = (10101010, 01010101,
        11001100, 00110011,
        11110000, 00001111,
        00000000, 11111111, random)

for each pattern
        overwrite file
remove file
```

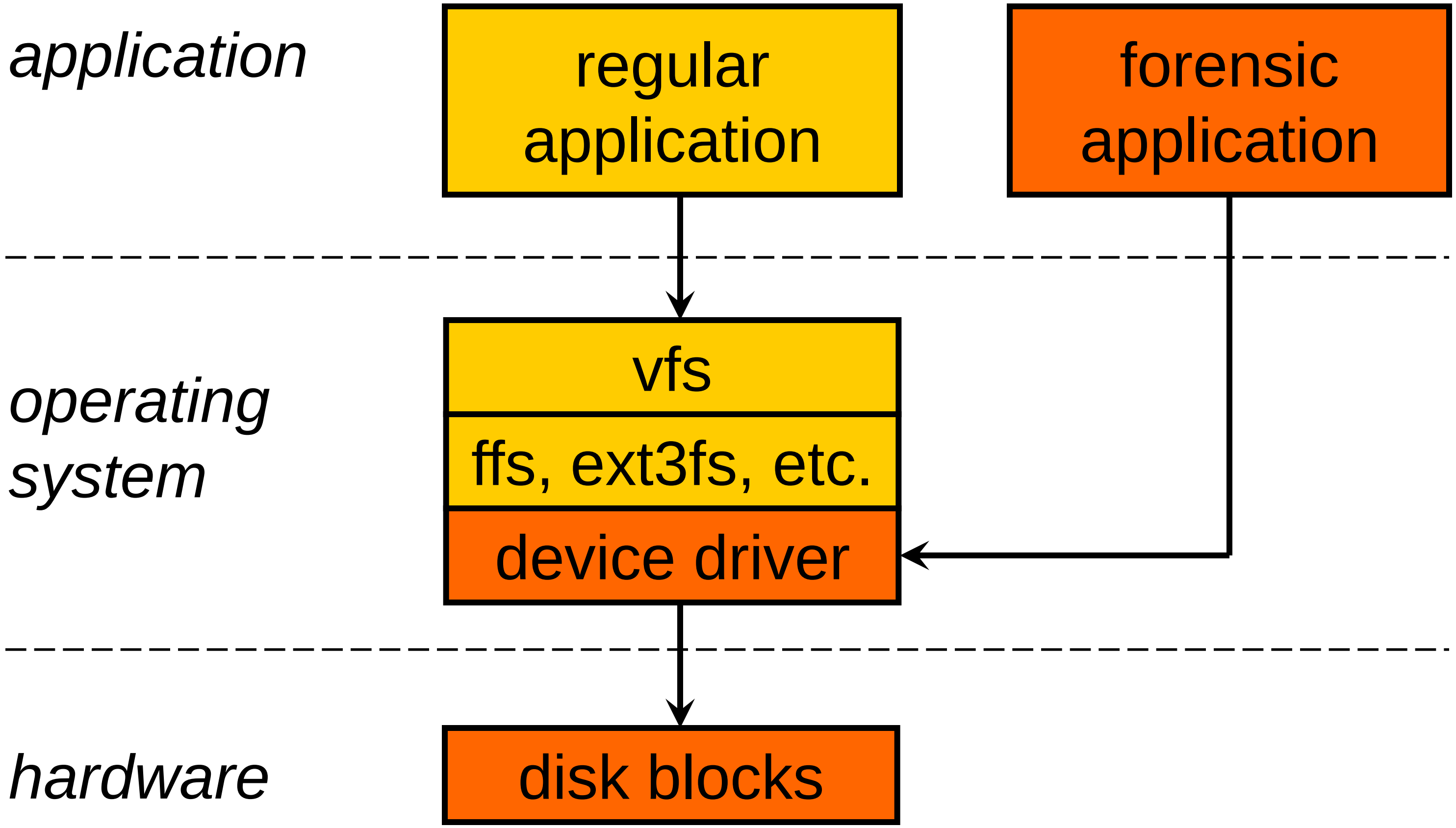
File shredder code, paraphrased

```
long overwrite(char *filename)
{
    FILE *fp;
    long count, file_size = filesize(filename);

    if ((fp = fopen(filename, “w”)) == NULL)
            /* error... */
    for (count = 0; count < file_size; count += BUFFER_SIZE)
            fwrite(buffer, BUFFER_SIZE, 1, fp);
    fclose(fp); /* XXX no error checking */

    return (count);
}
```

What can go wrong?

- The program fails to overwrite the target file content multiple times.
- The program fails to overwrite the target at all.
- The program overwrites something other than the target file content.
- Guess what :-).

Forensic tools to access (deleted) file information



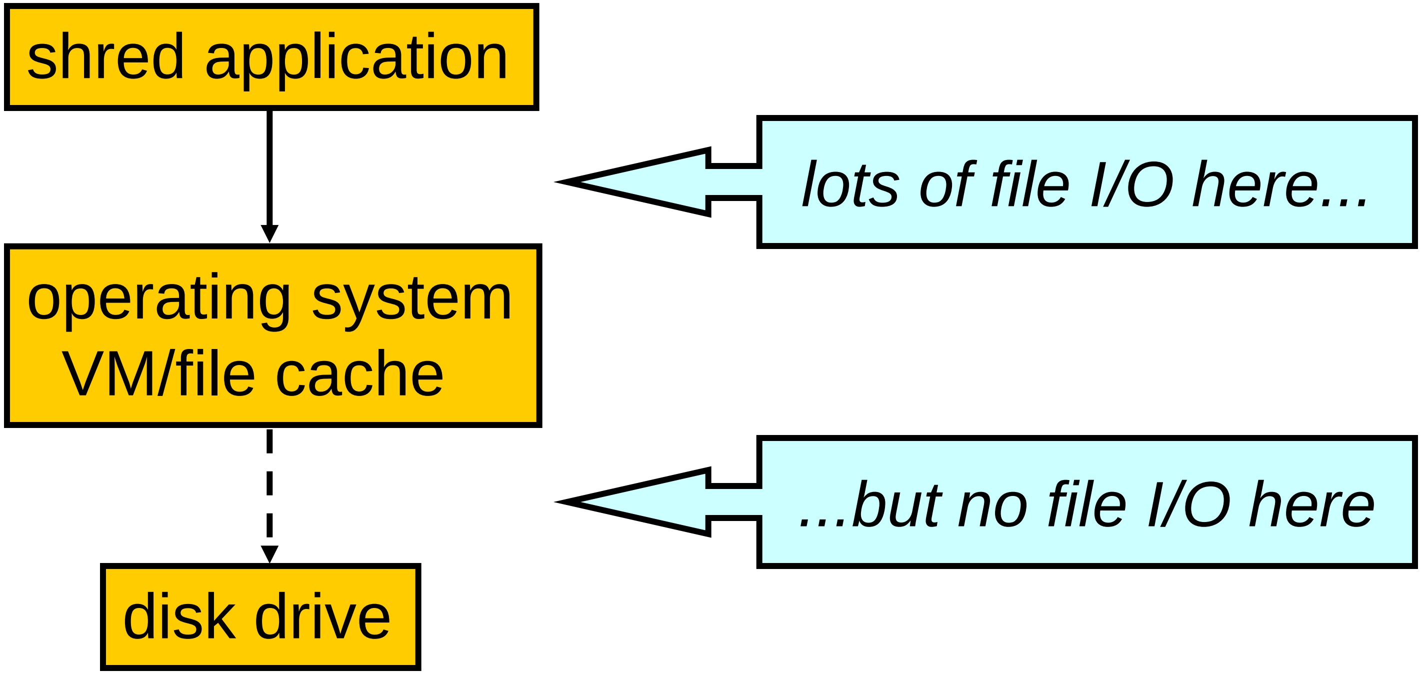
Coroner’s Toolkit discovery

(Note: details are specific to the RedHat 6 implementation)

```
[root test]# ls -il shred.me                        list the file with its file number
1298547 -rw-rw-r--   1 jharlan  jharlan      17 Oct 10 08:25 shred.me
[root test]# icat /dev/hda5 1298547                 access the file by its file number
shred this puppy
[root test]# shred shred.me                         overwrite and delete the file
Are you sure you want to delete shred.me? y
1000 bytes have been overwritten.
The file shred.me has been destroyed!
[root test]# icat /dev/hda5 1298547                 access deleted file by its number
shred this puppy                                    the data is still there!

[root test]#
```

See: http://www.securityfocus.com/archive/1/138706 and follow-ups.

Delayed file system writes



File shredder problem #1
Failure to overwrite repeatedly

- Because of delayed writes, the shred program repeatedly overwrites the *in-memory* copy of the file, instead of the *on-disk* copy.

```
for each pattern
          overwrite file
```

File shredder problem #2
Failure to overwrite even once

- Because of delayed writes, the file system discards the *in-memory* updates when the file is deleted.
- The *on-disk* copy is never even updated!

```
for each pattern
        overwrite file
remove file
```

File shredder problem #3
Overwriting the wrong data

- The program may overwrite the wrong data blocks. *fopen(path,”w”) truncates* the file to zero length, and the file system may allocate *different* blocks for the new data.

```
if ((fp = fopen(filename, “w”)) == NULL)
        /* error... */
for (count = 0; count < file_size; count += BUFFER_SIZE)
        fwrite(buffer, BUFFER_SIZE, 1, fp);
fclose(fp); /* XXX no error checking */
```

“Fixing” the file shredder program

```
if ((fp = fopen(filename, “r+”)) == 0)          open for update, not truncate
        /* error... */
for (count = 0; count < file_size; count += BUFFER_SIZE)
        fwrite(buffer, BUFFER_SIZE, 1, fp);
if (fflush(fp) != 0)                            application buffer => kernel
        /* error... */
if (fsync(fileno(fp)) != 0)                     kernel buffer => disk
        /* error... */
if (fclose(fp) != 0)                            and only then close the file
        /* error... */
```

Limitations of file shredding

- Write caches in disk drives and/or disk controllers may ignore all but the last overwrite operation.

- Non-magnetic disks (flash, NVRAM) try to avoid overwriting the same bits repeatedly. Instead they create multiple copies of data.

- Not shredded: temporary copies from text editors, copies in printer queues, mail queues, swap files.

- Continued...

Limitations of file shredding (continued)

- File systems may relocate a file block when it is updated, to reduce file fragmentation.
- Disk drives relocate blocks that become marginal.
- Journaling file systems may create additional temporary copies of data (ext3fs: journal=data).
- Copy-on-write file systems (like Solaris ZFS) never overwrite a disk block that is “in use”.
- None of these limitations exist with file systems that encrypt each file with its own secret key.

Lessons learned

- Step outside the high-level illusions that systems create for users and developers.
  - Optimizations in operating systems and in hardware may invalidate a program completely.
- Don’t assume, verify. Intruders don’t play by the rules of APIs or protocols.
  - Examine raw disk blocks (network packets, etc.)
- Are we solving the right problem? Zero filling all free disk space (and all swap!) may be more effective.