Software Footprints
Lies, damned lies and ...

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

Software footprints

- What files to look at (forensics, auditing)
- What programs run on a system
- What files those programs access (data, logs)
- What does it take to get a system into its current state
- Complexity: number of files, amount of code and data

Measuring a software footprint

- Kernel-level auditing support (yeah right)
- User-level call tracing (trace, strace, truss)
- Mactimes: file modification/access/change times
- Code: static analysis (strings)

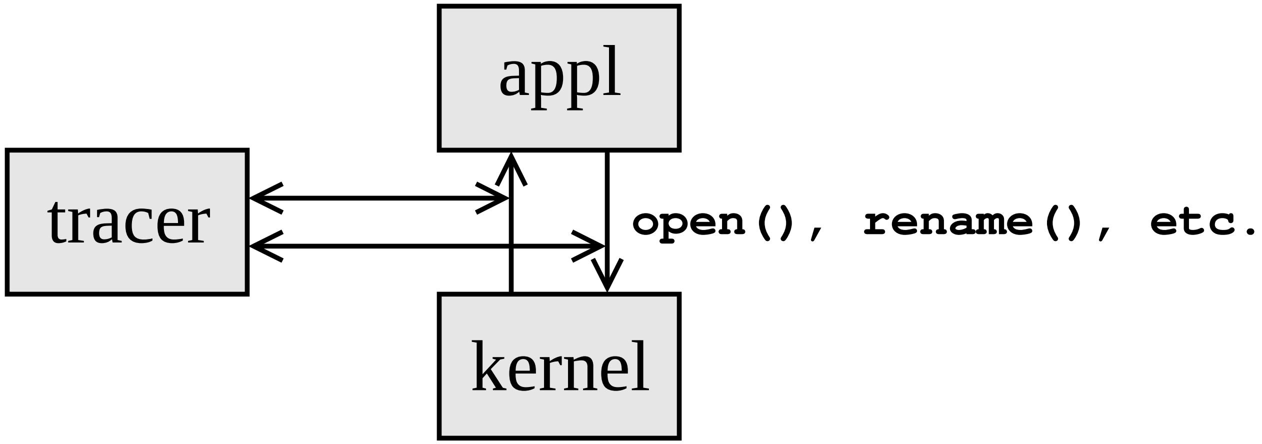
Static code analysis

- Look for pathnames in scripts
- Look for pathnames in source code
- Look for pathnames in binary programs
- Result: a large number of potential file access
- Problems: PATH, dynamic names, false positives

File MACtimes (modification/access/change)

- Show after-the-fact read/write/status change
- Do not show access frequencies
- Do not show what files were removed
- Do not show what pathname was used (hard/soft links)
- Do not work on read-only file systems (system boot)

User-level system call tracing



- Use process debugging hooks (ptrace(2) or /proc)
- trace (SunOS 4), truss (Solaris 2), strace (portable)
- BSD tracing sucks: ktrace(2) wants filename argument

VMware - a godsend

- Best of all, it actually works
- Multiple OSes in one box
- Real networking
- Undoable file systems
- Mediocre performance

VMware performance

- Simulated console needs tweaked graphics software
- Networks logins perform OK
- Postfix compile time (VMware 1.1.2, i686, 450MHz)

FreeBSD 3.3	134 sec (50 sec running natively)
RedHat 6.1	215 sec (51 sec running natively)
Solaris 7.0	479 sec (OS is not supported)
OpenBSD 2.5	583 sec (OS is not supported)

- Not yet good enough to replace a rack of old PCs

General sequence of UNIX system boot

- Load sequence of boot programs from disk (tiny-small)
- Load kernel from disk (large)
- Mount root file system read-only
- Execute /sbin/init
- Execute commands in /etc/inittab

Tracing the UNIX boot process

- Use available tracing facility (truss, strace)
- Tracing the init process is problematic
- Boot environment: read-only, no /proc
- Trace the commands run from /etc/inittab

Tracing detail

- Fragment from modified Solaris inittab file:

```
fs::sysinit:ftrace -o /var/tmp/ftrace-rcS /sbin/rcS

s0:0:wait:/sbin/rc0

s1:1:wait:/usr/sbin/shutdown -y -iS -g0

s2:23:wait:ftrace -o /var/tmp/ftrace-rc2 /sbin/rc2

s3:3:wait:ftrace -o /var/tmp/ftrace-rc3 /sbin/rc3

s5:5:wait:ftrace -o /var/tmp/ftrace-rc5 /sbin/rc5

s6:6:wait:/sbin/rc6
```

- ftrace runs truss or strace from the background

Boot-up system footprint, unique file counts

program	other	non-exist	
81	290	217	Redhat 4.1
86	494	289	Redhat 5.2
76	639	262	Redhat 6.1
65	250	229	Solaris 2.5.1
77	344	273	Solaris 7.0

NB "bare system" boot up without GUI

bare-files.fig

Boot-up system footprint, user-level code

program	library	total	
1.8M	1.6M	3.4M	Redhat 4.1
3.0M	4.4M	7.4M	Redhat 5.2
2.4M	7.6M	10M	Redhat 6.1
2.0M	1.8M	3.8M	Solaris 2.5.1
3.1M	3.5M	6.6M	Solaris 7.0

Systems double in only a few releases!

Default GUI login footprint, unique file counts

program	other	non-exist	
76	617	259	Redhat 6.1 non-GUI boot
107	1667	1090	Same, with GUI login/logout
77	344	273	Solaris 7.0 non-GUI boot
150	1153	1986	Same, with GUI login/logout

It is amazing that GUIs aren’t slower than booting!

Default GUI login footprint, user-level code

program	library	total	
2.4M	7.6M	10M	Redhat 6.1 non-GUI boot
6.1M	14.9M	21M	Same, with GUI login/out
3.2M	3.4M	6.6M	Solaris 7.0 non-GUI boot
6.1M	14.9M	21M	Same, with GUI login/out

Conclusion

Software bloat is universal in all dimensions

- Vertically:

  Kernels, libraries, and applications bloat equally

- Horizontally:

  Different vendors bloat equally

- The "Big Bang" of software engineering