Examining a process without disturbing it

TCP Wrapper-style alert

- Suspicious activity at some unlikely hour:

```
    Feb 13 23:09:52 wsbs06 in.fingerd[15900]:
                                    connect from lock@wsbs03
```

- Screen saver accounts don’t finger around at midnight.

- An intruder has compromised the screen saver account
  . . . and possibly more.

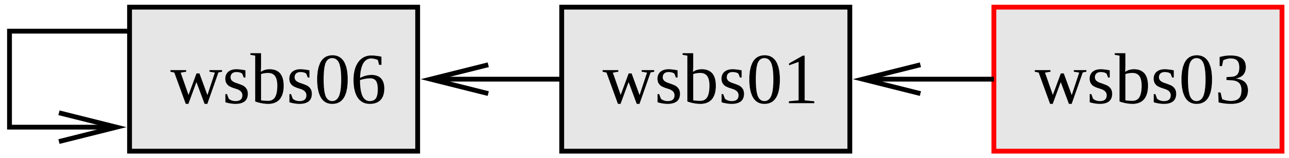
.

Bad news - compromised machine

- Someone compromised the root account on host wsbs03.

```
Feb 13 23:05:34 wsbs01 in.fingerd[7948]:
                       connect from root@wsbs03
Feb 13 23:05:35 wsbs06 in.fingerd[15895]:
                       connect from nobody@wsbs01
Feb 13 23:05:36 wsbs06 in.fingerd[15897]:
                       refused connect from nobody@localhost
```

- Someone is a finger-command virtuoso.



```
finger @localhost@wsbs06@wsbs01
```

Evidence of suspicious process

```
# ps aux
USER        PID %CPU %MEM   SZ   RSS TT STAT START   TIME COMMAND
root          0  0.0  0.0    0     0 ?  D    Jan 14  0:01 swapper
root          1  0.0  0.0   52     0 ?  IW   Jan 14  0:00 /sbin/init -
root          2  0.0  0.0    0     0 ?  D    Jan 14  0:00 pagedaemon
root         75  0.0  0.0   16     0 ?  I    Jan 14  0:00  (biod)
root         55  0.0  0.0   68     0 ?  IW   Jan 14  0:00 portmap
. . .
root      12823  0.0  0.0    48    0 ?  IW   23:02   0:00 <defunct>
. . .
```

- Process start time: matches time of incident.
- Process name: misleading to hide real purpose.

ps incantations (BSD-ish UNIX)

- "ps ax" for basic listing.

```
PID TT STAT  TIME COMMAND
. . .
152 p0 S     0:00 -csh (csh)
883 p0 R     0:00 ps ax
. . .
```

- "ps auxeww" for command, environment, and more.

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START   TIME COMMAND
. . .
wietse     152  0.0  1.5   56  212 p0 S    09:12   0:00 -csh
HOME=/home/wietse USER=wietse LOGNAME=wietse PATH=/bin:/usr/bin:
/usr/ucb:/usr/bin/X11:/usr/local/bin:/usr/local/bin
MAIL=/var/spool/mail/wietse SHELL=/bin/csh TERM=xterm (csh)
. . .
```

.

Will the real ps command stand up?

- System V-ish UNIX: "ps -ef" for minimally-useful listing.

```
     UID    PID   PPID  C    STIME TTY      TIME COMD
. . .
wietse  9157  9154 24 12:57:58 pts/0    0:00 -csh
wietse  9184  9157 21 13:00:43 pts/0    0:00 ps -ef
. . .
```

- System V-ish UNIX: "ps -ealf" gives marginally more.

```
 F S      UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN    STIME TTY      TIME COMD
. . .
 8 S   wietse  9157  9154 25  41 20 fc52bcc0    218 fc52be90 12:57:58 pts/0    0:00 -csh
 8 O   wietse  9204  9157 21  55 20 fc52b000    173          13:13:03 pts/0    0:00 ps -ealf
. . .
```

.

lsof - list open files, connections etc.

- Source: ftp://vic.cc.purdue.edu/pub/tools/unix/lsof

```
# lsof -p 12823
COMMAND      PID     USER   FD    TYPE     DEVICE   SIZE/OFF  INODE NAME
<defunct> 12823     root   cwd   VDIR      7,  22       1024 868362 /var
<defunct> 12823     root   T00   VREG      7,  22      32768 868676 /var
<defunct> 12823     root   T01   VREG      7,   6      24576 139429 /usr
<defunct> 12823     root   T02   VREG      7,   6     516096 139397 /usr
<defunct> 12823     root   T03   VREG      7,   0       4096  14951 /
<defunct> 12823     root   T04   VREG      7,   6      40960 139492 /usr
<defunct> 12823     root     3u  inet 0xff64b50c        0x0    TCP *:5120
```

- Something is accepting connections on TCP port 5120.

- No executable file found (find /var -inum 868676 -print).

Freezing the process

- Don’t connect to the port - bad things might happen.
- Don’t terminate the process - all info would be lost.
- Suspend the process until we have figured out what it is:

```
# kill -STOP 12823
```

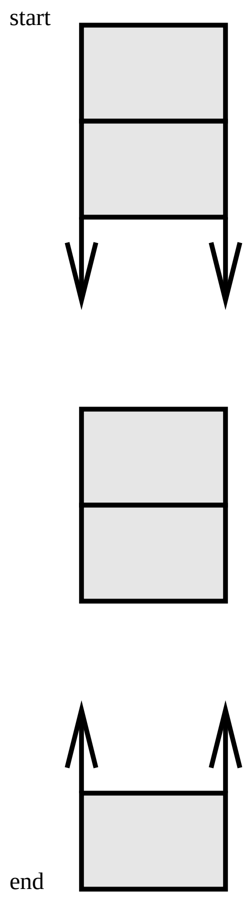
- Checking the result reveals yet another surprise.

```
# ps ax|grep T
  167 ?   TW     0:11 cron
12823 ?   TW     0:00 <defunct>
```

(cron is suspended, too)

Capturing process memory - intro

Simplified typical process memory map, not drawn to scale



program code and constants (from program file)

program variables (saved in core dump)

direction of growth of data segment

shared library code and constants (from lib. files)

shared library variables (saved in core dump)

direction of growth of stack segment

stack (saved in core dump)

Capturing process data - gcore

- Create "core dump" checkpoint of variables and stack.
- Example: core dump checkpoint of process 12832

```
# gcore 12832
gcore: core.12832 dumped
# ls -l core.12832
-rw-r--r--  1 root      8421808 Feb 24 09:29 core.12832
```

- Result can be examined with standard debugger tools, given copies of the program and shared library files.
- Result can be examined with unstructured tools such as "strings", binary editors, etc.
- gcore is not available on LINUX (but alternatives exist).

Capturing process info - /proc

- Entries in /proc/<pid> give access to process info.

Solaris	FreeBSD	LINUX	what
`object/a.out`	`file`	`exe`	program file
`as`	`mem`	`mem`	process memory
`map`	`map`	`maps`	memory map
`...`	`...`	`...`	etcetera...

- Capturing the program file is as simple as copying /proc/<pid>/file (or whatever they call it today).
- Capturing process memory requires more work because the memory map has holes in it (see the "pcat" utility).

Capturing the program file - icat

- icat - retrieve file, given device name and inode number.
- Recover deleted but still open or running files.
- Part of the software developed for this class.
- Example: save contents of file 868676 on /dev/sd2g.

  ```
  # icat /dev/rsd2g 868676 >868676.out
  ```

- Result can be examined with standard debuggers and with unstructured tools such as "strings", binary editors, etc.

Capturing process memory - pcat

- Dump the entire memory of a process to file - including code, data, heap, libraries and stack.
- Part of the toolkit developed for this class.
- Example: dump all memory of process 12832.

```
# pcat 12832 >pcat.12832
```

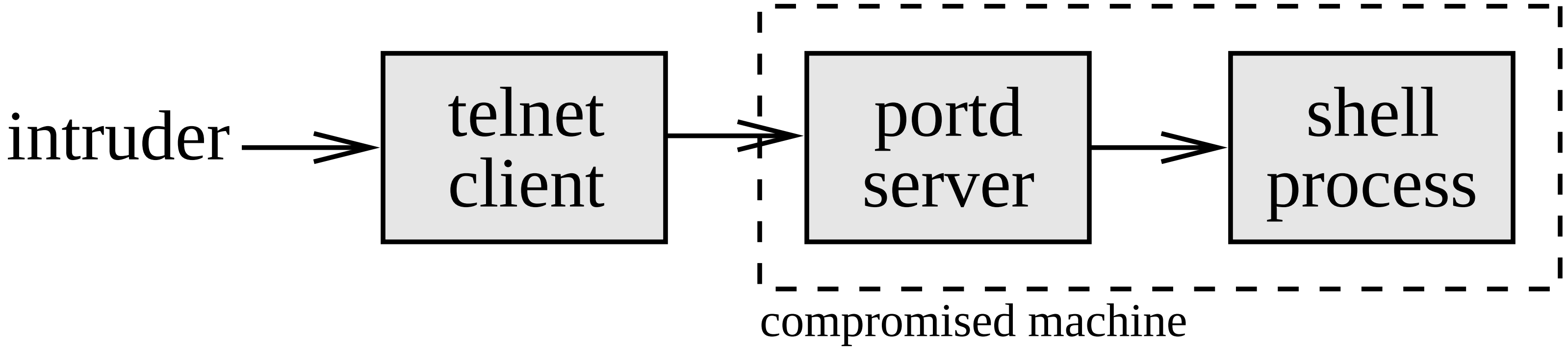
- Result can be examined with unstructured tools such as "strings", binary editor, etc.

First examination with "strings"

```
# strings core.12832 | more
..stuff...
Error: cant open file
kill
Error: cant open file
%s not found
bad port %s
Trying %s...
telcli: socket
:) %s port %d...
csh -bif
exec
pqrstuvwxyzPQRST
/dev/ptyXX
/dev/pty
/dev/ptyp
0123456789abcdef
/bin/csh
/dev/
/dev/tty
fork
/bin/csh
telnetd: %s.
...more stuff...
```

Backdoor service (portd variant)

- Stand-alone telnet server.
- Bypass TCP Wrapper and system login procedure.

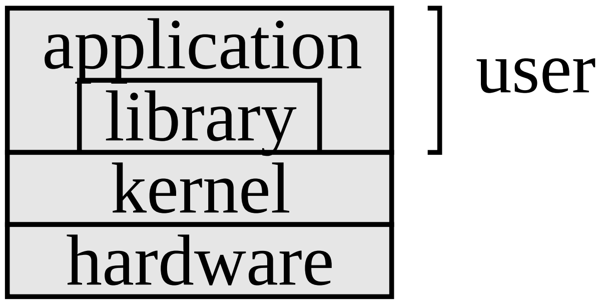


```
% telnet victim 5120
Trying 131.155.210.17...
Connected to victim.
Escape character is '^]'.
password

SunOS UNIX (victim)

victim#
```

Watching a process in action - intro

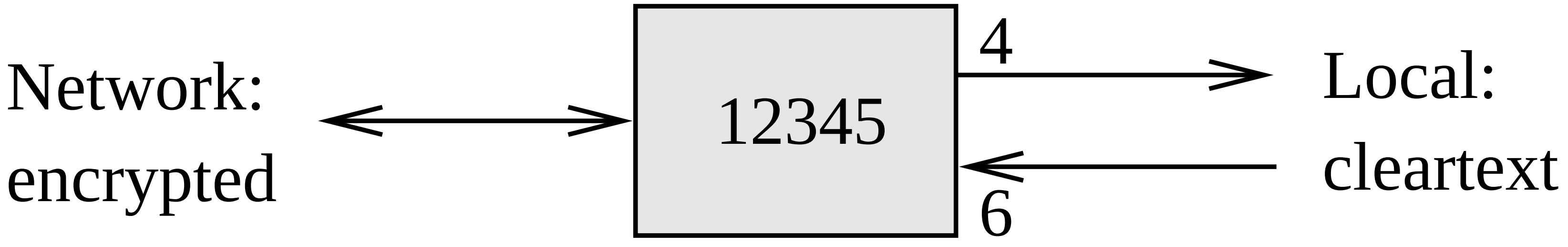


- Use standard debugging hooks to intercept and log:
  - System calls (tapping the user-kernel interface).
  - Library calls (tapping the application-library interface).
  - Individual application routines (requires program file).
  - Individual machine instructions.
- Run-time tracing can generate large amounts of data.
- Run-time tracing can impact performance noticeably.

Watching system calls

- User-kernel interface: does not show what happens inside the application or inside library routines.
- All information must enter or leave the program via a system call: input, output, network, file, terminal, etc.
- Many system-specific tools: trace (SunOS 4), truss (Solaris 2), ktrace (*BSD), etc.
- Portable system call tracer: strace, originally by Paul Kranenburg, ported and extended by many.

Syscall tracing to decrypt traffic



`# strace -p 12345`	watch process 12345
`-f`	and its child processes
`-e trace=read,write`	look at read/write calls only
`-e read=6`	show everyting read from ch. 6
`-e write=4`	show everything written to ch. 4

.

Other host-based tracing

- ltrace: log every library routine call (output like strace).
  ftp://ftp.debian.org/debian/dists/unstable/main/source/utils/
  Guaranteed portable to LINUX.

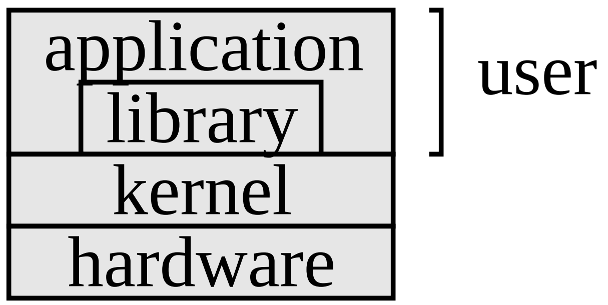
- ttywatcher: real-time monitoring and more.
  ftp://coast.cs.purdue.edu/pub/tools/unix/ttywatcher/
  tap: hook into streams-based tty systems.
  ftp://coast.cs.purdue.edu/pub/tools/unix/tap/
  Guaranteed portable to SUNs.

- The uncensored logdaemon utilities.

Hiding a process from observation

- Standard B2+ security feature (covert channels).
- Otherwise, retrofitted by hacking system software.
- Spying on an intruder without being seen.
- Hiding a password sniffer process.
- Other forms of surveillance.

Hiding a process from observation



- Modified ps/lsof/top/etc. applications and/or library routines. Can be sufficient when process listing applications must be installed as privileged commands.

- Modified kernel: crude implementations based on loadable kernel modules from http://thc.pimmel.com/ Hides a process even from the most privileged users.

- Can’t use lots of CPU, memory or I/O resources.