Introduction to Quick and Dirty Forensics

Vincent Brillault, originally written by Leif Nixon

SO, YOU THINK YOU MAY HAVE AN INCIDENT?

- Monitoring alarm
- External alert
- Anomalous system behaviour

IS THIS FOR REAL? – INCIDENT TRIAGE

Look at things like:

- system logs
- command line histories
- `ps`
- `top`
- `netstat`
- `lsof`
- ...

Do *not*:

- run `rpm -Va`
- reboot the system
- kill suspect processes
- delete malicious files
- ...

OBSERVATION CHANGES THE OBJECT

- List files in a folder: Update (folder) timestamp
- Read a file: Update timestamp
- Run a command: Update timestamp
- Write data: Override *free* sectors
- Actions while *syslog is running: Override *free* sectors

Do the *least intrusive* investigation possible.

OH, SH⋆T!

It's real! You've been hacked!

Quick, what's the first thing you do?

OH, SH⋆T!

It's real! You've been hacked!

Quick, what's the first thing you do?

Stop!

Take a break. Go have a cup of coffee. Or tea. Or a can of soda. (Beer is probably not a good idea, though.)

IS THIS REALLY YOUR PROBLEM?

- Where do you want to go with it?
- Do you want *evidence* or *leads*?

THE LEGAL ROUTE

- Can you get enough data to go to a trial?
- Do you have a policy requiring legal investigations?

$\Rightarrow$ Don't touch it, let officials do it

$\Rightarrow$ Sorry, this presentation is not for you....

THE ALTERNATIVE ROUTE

- You won't be able to identify actors?
- Your policies forbid you to involve law enforcement?

$\Rightarrow$ You just wand to get back into service, right?

$\Rightarrow$ Can you simply reinstall the host?

THE ALTERNATIVE ROUTE

- You won't be able to identify actors?
- Your policies forbid you to involve law enforcement?

$\Rightarrow$ You just wand to get back into service, right?

$\Rightarrow$ Can you simply reinstall the host? *Nooo!*

OUR GOAL

To get back into *secure* service we would like to know:

- How the intruders got in
- When they did so
- What they have been doing on the system
- What we can do to stop them from returning
- Which other sites that may have been hit

$\Rightarrow$ Otherwise they will come back!

WE MUST TALK ABOUT THIS

One extremely important task during an incident:

Talk to people!

- Victims
- Users
- Management
- Partner sites
- Other security teams

⇒ Teams must have one coordinator for communications

QUICK AND DIRTY FORENSICS

- *Proper* forensics: careful & thorough
  - Hard to perform
  - Takes a long time
- *Quick & dirty* forensics: might be good enough or even *better*

MAKE SURE YOU ARE NOT INTERRUPTED

- Check open network connections/listening ports (netstat)
- Isolate the system:
  - Unplug the Network cable
  - Add router/firewall
  - Change existing router/firewall rules
- Virtualization: use snapshots!

DATA TYPES VOLATILITY

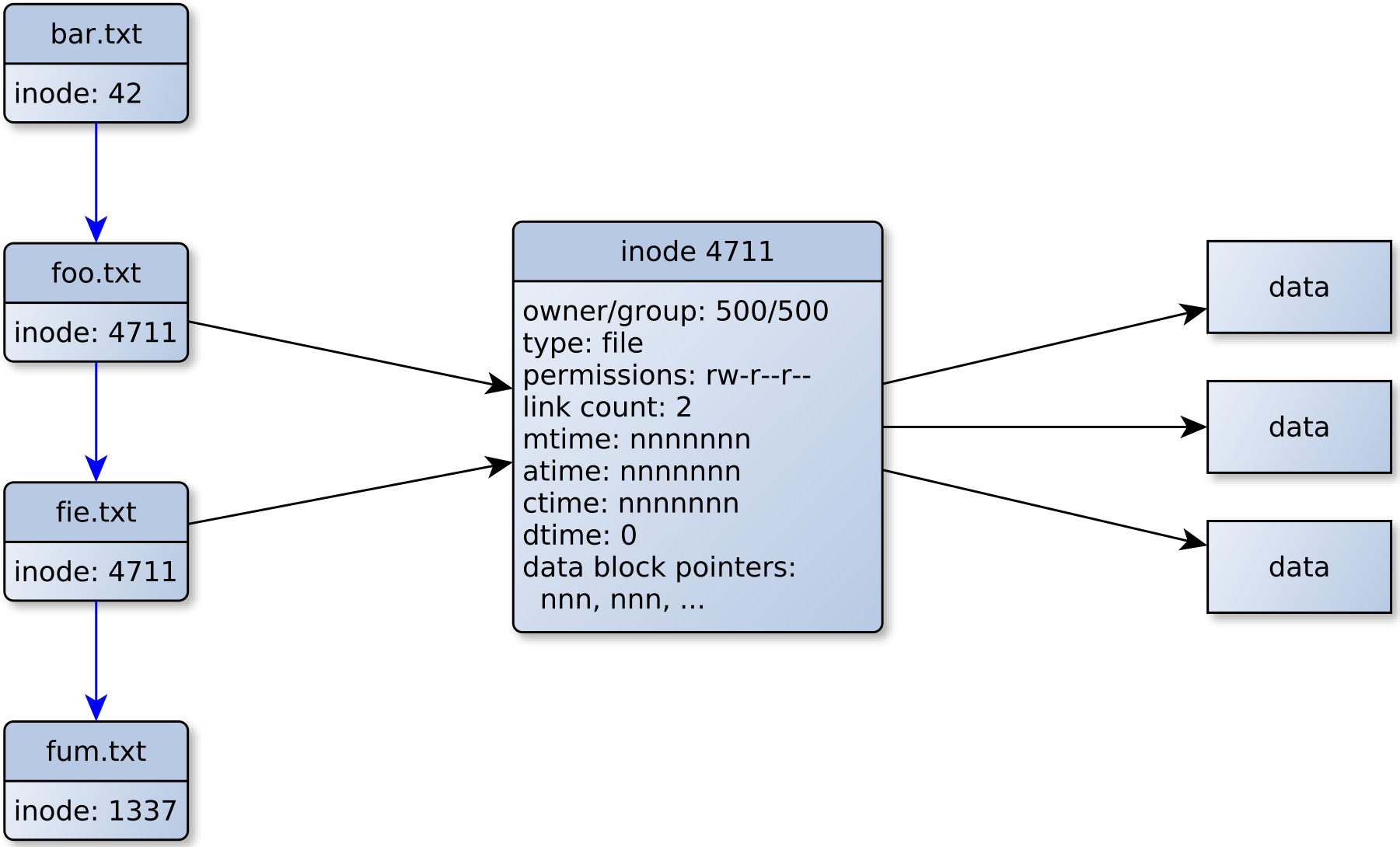
Registers, peripheral memory, caches, etc.	nanoseconds
Main memory	nanoseconds
Network state	milliseconds
Running processes	seconds
Disk	minutes
Backup media, etc.	years
Printouts, etc.	tens of years

(Table borrowed from "Forensic Discovery", Farmer & Venema, Addison-Wesley 2005). You should buy this book.)

THE ORDER OF VOLATILITY

- Follow volatility order when collecting data
- Over-prioritize file-system timestamps (details later)

SOMETHING ABOUT FILESYSTEMS



TYPES OF TIMESTAMPS

mtime – modification time; the last time the *contents* (data blocks) of a file changed

atime – access time; the last time the file was read[1]

ctime – change time; the last time the file contents or one of the attributes in the inode changed

dtime – deletion time; recorded in deleted inodes (ext*n*fs)

crtime – creation time; ext4fs only

[1] Death to the `noatime`/`relatime` mount options!

TRUSTWORTHINESS OF TIMESTAMPS

All timestamps, like any cold data can be manipulated:

- Using direct commands: atime & mtime via *touch*
- Setting the server back in time: ctime & crtime
- Raw modifications on disks[2]: ctime & crtime

⇒ Always question your findings, try to corroborate:

- Check inode sequences
- Check incoherences (file–folder relations)
- Check file system journal
- Correspondence with external activity

[2] Which is rather tricky if the disk is mounted

GENERATING TIMELINES FROM TIMESTAMPS

- Collect timestamp from entire filesystem
- Sort them: timeline of past events
- Two ways:

  From within: `stat` every file in the mounted filesystem

  From outside: Dig into the raw device/image with specialized tools

TIMELINES THE QUICK AND DIRTY WAY

- Collect data: one simple line
- Generate timeline: sort & make it readable

```
find / -xdev -print0 | xargs -0 stat -c "%Y %X %Z %A %U %G %n"
>> timestamps.dat

timeline-decorator.py < timestamps.dat | sort -n > timeline.txt
```

TIMELINES THE QUICK AND DIRTY WAY

timeline-decorator.py:

```
#! /usr/bin/python

import sys, time

def print_line(flags, t, mode, user, group, name):
    print t, time.ctime(float(t)), flags, mode, user, group, name

for line in sys.stdin:
    line = line[:-1]
    (m, a, c, mode, user, group, name) = line.split(" ", 6)
    if m == a:
        if m == c:
            print_line("mac", m, mode, user, group, name)
        else:
            print_line("ma-", m, mode, user, group, name)
            print_line("--c", c, mode, user, group, name)
    else:
        if m == c:
            print_line("m-c", m, mode, user, group, name)
            print_line("-a-", a, mode, user, group, name)
        else:
            print_line("m--", m, mode, user, group, name)
            print_line("-a-", a, mode, user, group, name)
            print_line("--c", c, mode, user, group, name)
```

TIMELINES THE QUICK AND DIRTY WAY

`stat` all files

- Really easy, no tool, no prerequisite
- Be sure to store output on external storage
- Change the filesystem state! (atime)
- Miss deleted files
- Fooled by rootkits

SLIGHTLY SLOWER AND CLEANER TIMELINES

The Sleuth Kit[3], TSK

- opensource toolkit to manipulate disk device/images
- A bit more complex to use

```
fls -r -m / /dev/sda1 > rootfs.body
mactime -b rootfs.body > rootfs.timeline
```

[3] http://www.sleuthkit.org/

TSK TIMELINES

The Sleuth Kit[4], TSK

- Bypass operating system: reveals hidden files
- Lists deleted files
- A bit harder to use
- Need TSK binaries:
  - Compile them in place
  - Transfer them from outside
  - Use an external storage (NFS, USB stick...))
  - Work remotely on an image of the disk

[4] http://www.sleuthkit.org/

EXAMPLE

```
Tue Aug 16 2011 14:03:15 .a. r-xr-xr-x root    root    /usr/bin/w
Tue Aug 16 2011 14:03:28 .a. rwxr-xr-x root    root    /usr/bin/curl
Tue Aug 16 2011 14:03:36 .a. rwxr-xr-x root    root    /usr/bin/bzip2
Tue Aug 16 2011 14:04:41 .a. rwxr-xr-x root    root    /usr/bin/shred
Tue Aug 16 2011 14:06:26 .a. rw-r--r-- root    root    /usr/include/crypt.h
Tue Aug 16 2011 14:07:25 m.. rwxrwxr-x x_lenix x_lenix /var/tmp/...
Tue Aug 16 2011 14:08:01 m.c rw-r--r-- root    root    /var/tmp/.../openssh-5.2p1.tar.bz2 (deleted-realloc)
Tue Aug 16 2011 14:08:01 m.c rw-r--r-- root    root    /var/tmp/.../openssh-5.2p1 (deleted-realloc)
```

WHAT DOES THE TIMELINE TELL US?

- ctimes often tells us when files were created
- atimes can tell us when files were read and binaries executed
- mtimes can be useful *because* they can be modified

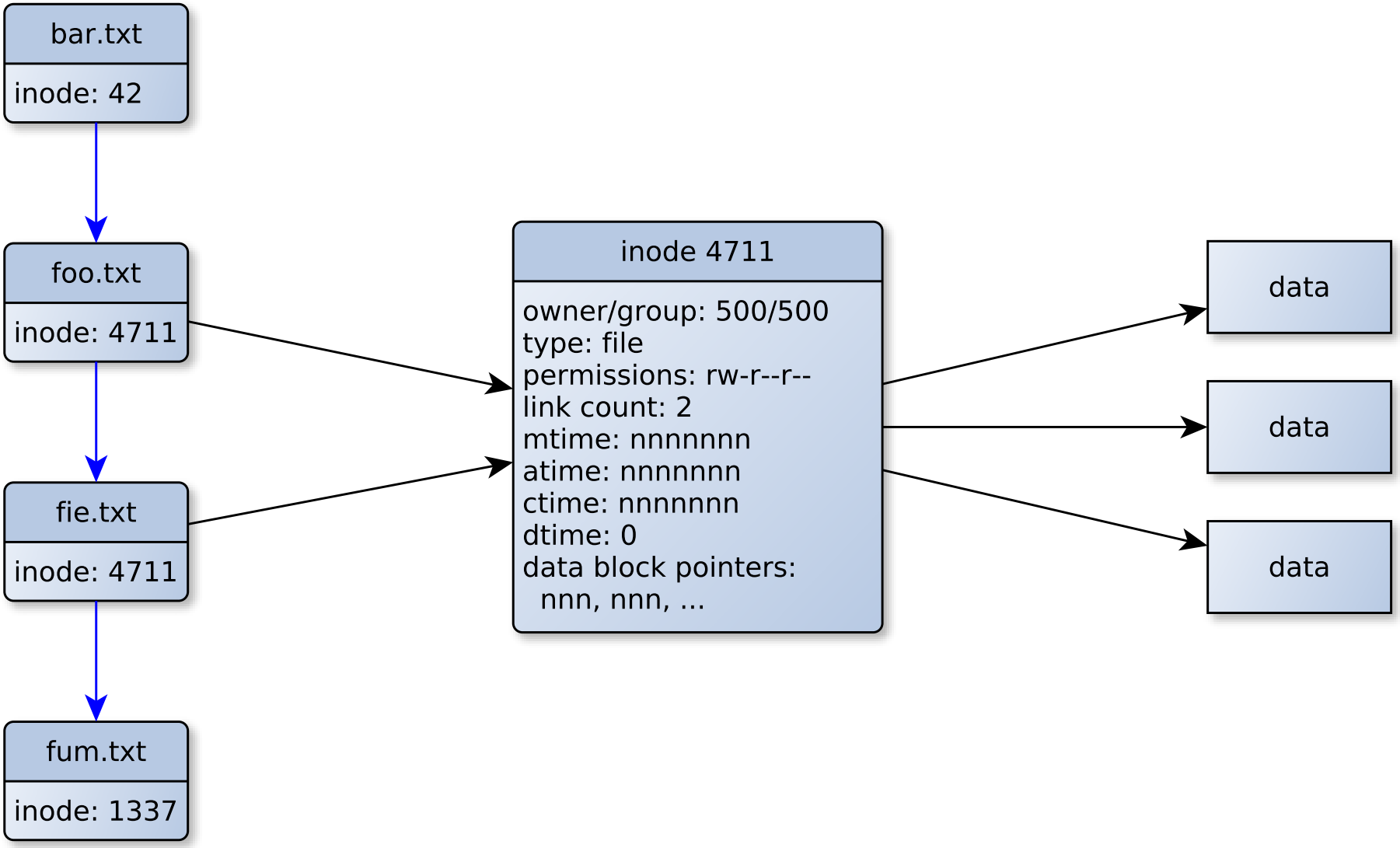
MTIME PRESERVATION

```
[nixon@host1]$ stat fie
  File: ‘fie’
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: fd01h/64769d Inode: 314704      Links: 1
Access: (0664/-rw-rw-r--)  Uid: (  500/   nixon)   Gid: (  500/   nixon)
Access: 2012-04-19 13:39:29.311321819 +0200
Modify: 2012-04-19 13:39:29.311321819 +0200
Change: 2012-04-19 13:39:29.311321819 +0200
 Birth: -

[nixon@host1]$ scp -p fie host2:

[nixon@host2]$ stat fie
  File: ‘fie’
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: ca10h/51728d Inode: 2013283013  Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 7090/   nixon)   Gid: ( 7090/   nixon)
Access: 2012-04-19 13:39:29.000000000 +0200
Modify: 2012-04-19 13:39:29.000000000 +0200
Change: 2012-04-19 14:15:50.905321991 +0200
```

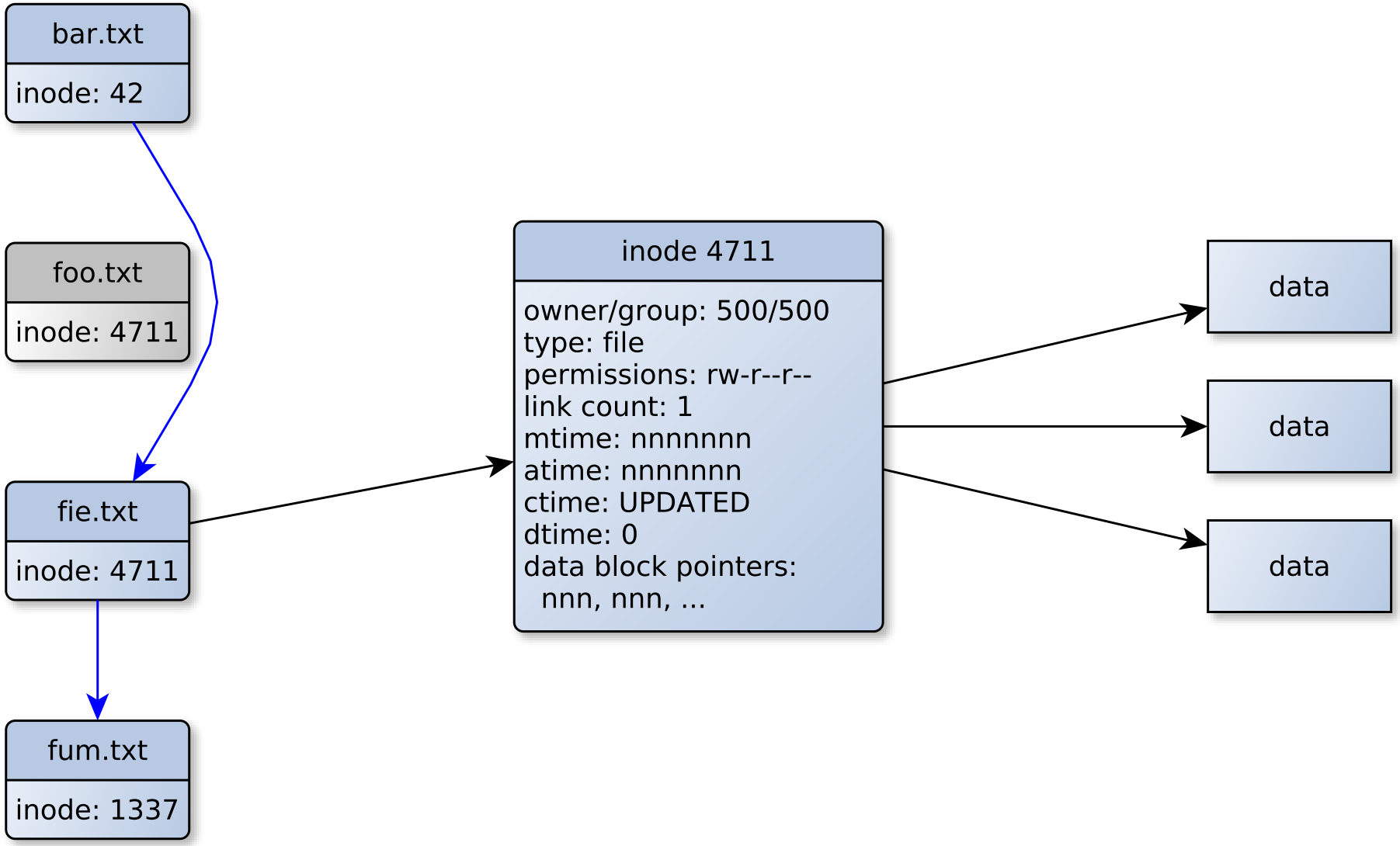
TRAPS AND COMPLICATING FACTORS

- You only see the last timestamp – easy to forget!
- Prelinking
- `updatedb`
- `tmpwatch`
- `makewhatis`

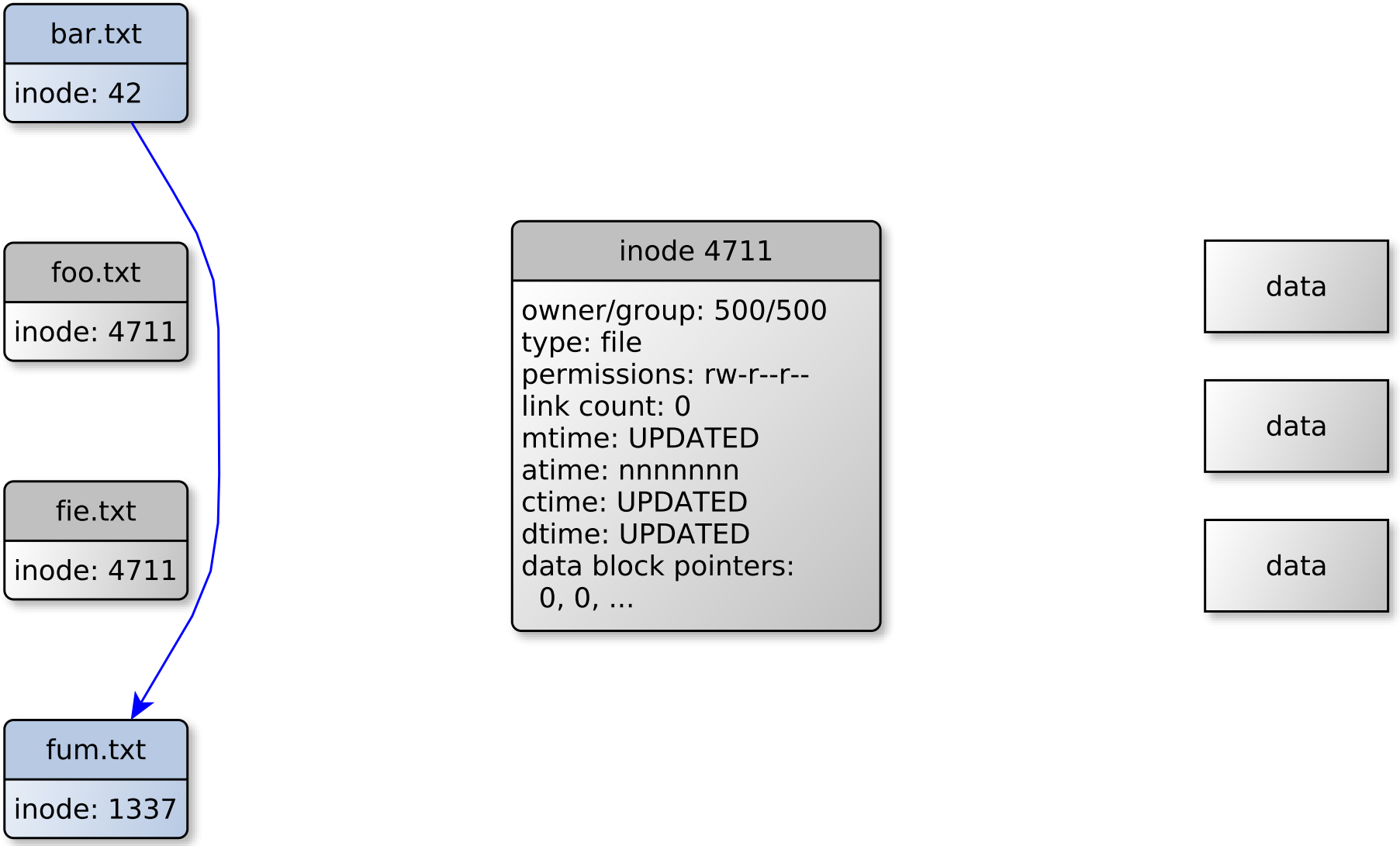
LOOKING AT DELETED DATA



LOOKING AT DELETED DATA



LOOKING AT DELETED DATA



LOOKING AT DELETED DATA

- Retrieving the inode is easy (TSK)
- If the file was recently deleted: journal
  - `extundelete`
  - `ext3grep`
- Data might still be on disk:
  - Tools like `photorec`
  - `grep`!

```
strings sda.img | grep "sshd.*Accepted "
```

LOOKING AT SLACK SPACE

- Slack space?
  - Disk space allocated in (typically) 4096 bytes blocks
  - Files rarely multiple of 4096

  $\Rightarrow$ The slack space is this remaining space
- Might contain old data from deleted files
- Might be used by intruder to hide data

$\Rightarrow$ Can be found via `grep`ing the all image

WORKING WITH DISK IMAGES

Grabbing a disk image is easy enough. To get the whole disk:

```
dd if=/dev/sda of=sda.img bs=512
```

Just a specific partition:

```
dd if=/dev/sda1 of=sda1.img bs=512
```

Caution: if disk is mounted at the time, the resulting image will be inconsistent and probably not mountable. Still, TSK will be able to work with it.

WORKING WITH DISK IMAGES

Listing and extracting partitions with TSK:

```
$ mmls -a sda.img
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
02:  00:00   0000002048   0000022527   0000020480   Linux (0x83)
06:  01:00   0000024576   0000126975   0000102400   Linux (0x83)
10:  02:00   0000129024   0000169983   0000040960   Linux Swap / Solaris x86 (0x82
14:  03:00   0000172032   0000262143   0000090112   Linux (0x83)

$ mmcat sda.img 6 > sda2.img

$ ls -lh
total 51M
-rw-rw-r-- 1 nixon nixon  50M Apr 22 12:36 sda2.img
-rw-rw-r-- 1 nixon nixon 129M Apr 22 12:32 sda.img
```

WORKING WITH DISK IMAGES

Images can be loopback-mounted for easy access:

```
mount -o loop,ro,noload sda2-copy.img mnt
```

Might fail:

- Corrupted image
- Need to replay the journal

$\Rightarrow$ Try remounting the image read-write temporaril[5]:

```
mount -o loop sda2-copy.img mnt
umount mnt
mount -o loop,ro sda2-copy.img mnt
```

[5] or by using the `norecovery` mount option, but then the filesystem may be inconsistent

WORKING WITH DISK IMAGES

To avoid time- and space-consuming copy operations, you can work with partitions in-place:

```
fls -o 24576[6] -r -m / sda.img > rootfs.body
```

```
mount -o ro,loop,offset=12582912[7] sda-copy.img mnt
```

[6] Offset in sectors, as reported by `mmls`

[7] Offset in bytes, i.e. 24576×512

LOOKING AT OTHER DATA

Root-compromised systems might lie to us!

- Use static-compiled binary from another system
- Use multiple tools, e.g. ps, pstree, top, ...
- Try to assess system by e.g. `rpm -Va`[8]

[8] *Don't* do this before you have gathered timestamps, since it will zap all atimes!

LOOKING AT PROCESSES

- Process names have no meaning (can be faked)
- Look for anomalies:
  - Duplicate system processes
  - Strange inheritances, e.g. `ping` should not have a `bash`
- Look at pid numbers: kernel/system pids are often in a narrow range
- Check /proc/id/exe

LOOKING AT OPEN FILES AND SOCKETS

- `netstat`
- `lsof`

$\Rightarrow$ Store it remotely if possible

LOOKING AT MEMORY

- Malicious process:
  - Dump it with `gcore`[9]
  - Use `strings` on it (can find e.g. domains, ips)
- Entire system: More difficult
  - Dumping: `https://code.google.com/p/lime-forensics/`
  - Analysis: Volatility `https://code.google.com/p/volatility/wiki/LinuxMemoryForensics`

[9] part of the gdb package

QUICK AND DIRTY MALWARE ANALYSIS

If you find a malicious binary, running `strings -a`[10] on it can yield interesting results.

You can also try to execute the binary under `strace` and `ltrace` to see in greater detail what it is doing. *This must be done very carefully*, preferably on an isolated host (like e.g. a VM without network access).

[10] Little known fact: `strings` will try to be smart when it's being run on an ELF binary and just look through certain ELF sectors. Use `-a` to read the whole file.

QUICK AND DIRTY MALWARE ANALYSIS

`ltrace` and `strace` of a suspect `sshd` binary when logging in as `myuser:mypassword`:

```
:
:
3348 strcmp("mypassword", ".ssh/authorized_keys2  ") = 1
3348 memset(0x7fff24742210, '\000', 2048)        = 0x7fff24742210
3348 memset(0x7fff24742c10, '\000', 512)         = 0x7fff24742c10
3348 memset(0x7fff24742a10, '\000', 512)         = 0x7fff24742a10
3348 strcat("SR: '", "myuser")                   = "SR: 'myuser"
3348 strcat("SR: 'myuser' '", "mypassword")          = "SR: 'myuser' 'mypassword"
:
:

:
:
3318  <... read resumed> "\n\0\0\0\6mypassword", 11) = 11
3321  read(4,  <unfinished ...>
3318  open("/usr/share/kbd/keymaps/i386/azerty/c1", O_RDWR|O_CREAT|O_APPEND, 0666)
3318  getuid()                                   = 0
:
:
```

OBFUSCATED DATA

Often, trojans will obfuscate strings (e.g. filenames) in the binary and data in log files. This is almost, almost always done by xor:ing the data with a single byte.

So, if a file contains binary junk, try xor:ing it with different values until you find something interesting.

OBFUSCATED DATA

```
$ file azerty/c1
azerty/c1: data

$ xor.py azerty/c1

$ ls
0x01.out  0x21.out  0x41.out  0x61.out  0x81.out  0xa1.out  0xc1.out  0xe1.out
0x02.out  0x22.out  0x42.out  0x62.out  0x82.out  0xa2.out  0xc2.out  0xe2.out
:
:

$ grep SR: *.out

0xff.out: SR: 'myuser' 'mypassword'
```

OBFUSCATED DATA

```
#! /usr/bin/python

import sys, argparse

def xor(buf, n):
    f = open("0x%02x.out" % n, "w")
    for c in buf:
        f.write(chr(ord(c) ^ n))
    f.close()

parser = argparse.ArgumentParser(description="xor a file with one or several integer values, output to one or
parser.add_argument("-n", help="Integer to xor with (default: loop over 1-255)")
parser.add_argument('infile', nargs='?', type=argparse.FileType('r'),
                    default=sys.stdin, help="Input file (default: stdin)")
args = parser.parse_args()
if args.n:
    if args.n.startswith("0x"):
        n = int(args.n, 16)
    else:
        n = int(args.n)
else:
    n = None

data = args.infile.read()

if n:
    xor(data, n)
else:
    for i in range(1,256):
        xor(data, i)
```

LOOKING AT LOGS

In most root intrustion, local logs will be sanitized

- This is not an issue as you are logging centrally, *right*?
- Otherwise, remember that different things go in different places:
  - `/var/log/secure` – ssh logs
  - `/var/log/wtmp` – binary db of terminal sessions
  - `/var/log/btmp` – binary db of failed logins
  - `/var/log/lastlog` – binary (sparse file) db of latest logins per user
  - `/var/log/audit/audit.log` – events from the audit subsystem
- Even if the intruder has tried to remove his traces, he might have missed one of these places – check them all!

A BRIEF NOTE ON ROOTKITS

The main problem with the quick and dirty approach to forensics is that we are placing a lot of trust in the tools on the system. If the intruder has deployed a rootkit, we may be in trouble.

USER LEVEL ROOTKITS

User level rootkits basically work by replacing key system binaries. These can often be discovered by running e.g. `rpm -Va` (this of course presumes that `rpm` itself is trustworthy – you may want to use several different methods to verify binaries).

KERNEL-BASED ROOTKITS

Kernel-based rootkits instead subverts the running kernel into lying about the state of the system. Kernel rootkits can typically hide the existence of certain files and processes. These rootkits can be hard to detect, but tools like `chkrootkit` and `rkhunter` can find some common kinds of rootkits.

It is also noteworthy that TSK usually can detect files hidden by kernel rootkits, since it bypasses most of the kernel filesystem stack.

PUTTING IT ALL TOGETHER

Once you have all your data, remember to compare and cross-check your data sources; Do command line histories match atimes on binaries? Do filesystem timestamps match login/out times? Etc, etc.

This not only increases the confidence level of your data, it can also help you gain new insights.

For example, if filesystem timestamps indicate attacker activities taking place at times when there were no active logins, perhaps there is a backdoor on the system that you have missed.

GOING BACK INTO SERVICE

After your forensic investigation, you hopefully have a pretty good idea of the intruder's actions. This allows you to clean your system from malware and to plug any hole the intruder might have exploited.

However, a *root compromised* system will almost always need to be installed. Still, your investigation will help you stop the same thing from happening in the future.

A QUICK AND DIRTY CONCLUSION

We have looked at some simple methods to collect forensic data. These methods are somewhat fragile and can be fooled by a clever attacker.

However, most attackers aren't very clever, and the quick and dirty approach surprisingly often can give a surprisingly detailed picture of the intruder's actions.

All sysadmins should know some basic quick and dirty forensics methods.

Hopefully, you do so now. Let's try it!