

# Performing a Forensic Investigation

By Joel Weise and Brad Powell

The following are the primary steps of the investigation as described in this section:

- 1 Isolate the system and data.
- 2 Collect information by interviewing all system administrators and anyone else who might have come into contact with the system.
- 3 Preserve evidence and create copies for analysis
  - ✍ Memory and /proc
  - ✍ Create a duplicate of the system
  - ✍ Validate copies.
- 4 Prepare for an analysis.
- 5 Perform an analysis.
- 6 Perform recovery.

## Isolate the System and Data

The purpose of the isolation step is to prevent the corruption of other systems and to ensure that you maintain the state of the system. By isolating the targeted system, you can reduce the opportunity of a cascading failure throughout an organization's IT infrastructure. The other purpose of isolating the system is to ensure an exact image of the system is preserved for possible prosecution. Isolation involves the following tasks:

- ✍ Remove the system(s) in question's network connectivity by physically pulling the network cables.
- ✍ Ensure that no one other than the forensic analyst touches anything. This includes everyone from the CEO and CTO, to maintenance personnel and interns.

This assumes that no hardware disk duplication device is available, which is the

preferred method, and duplication must be performed ad hoc.

## Collect Non-Technical Information

This step is performed to ensure you have good notes and can develop a timeline of events. The premise here is to start a log book using a fresh pad of paper to gather information from the system administration staff and any other persons involved. It is important to collect this information as soon as possible to ensure that the people involved don't forget things and to document the times and dates when the suspected break-in was detected. This information is critical in a courtroom and will also help narrow the search parameters when you start looking at the electronic data. If you can document the time and date when the suspected break-in occurred, you have a reference point that might prevent you from having to review logs from years past, thus narrowing search to the relevant time. This makes the analysis task much more manageable and improves the likelihood that you will discover when the initial penetration of the system occurred. At a minimum, document the following information:

- ✍ The time and date of when the suspected break-in occurred.
- ✍ The names, job functions, and contact information of all parties involved, including how they were involved.
- ✍ Times, dates, and names of when management got involved.
- ✍ Times and dates of investigation started and ended.
- ✍ Times, date names, and contact information of the people performing the investigation.
- ✍ Every step, process, and procedure performed on the data gathered and processed.

## Preserve Evidence and Create Copies for Analysis

The purpose of this process is twofold. You want to ensure that the original version of the attacked/corrupted system is copied and stored as evidence for possible future prosecution, and then you want to have a separate copy that can be used to search and analyze to figure out what happened and when. Before performing this task, ensure that the system has been disconnected from all other networks.

### **1 Mount (vold) the forensic CD or use NFS mount mini hub or Ethernet crossover cable to isolate the system.**

Do this so you can collect data on the running processes, perform `memdump(1)`, and copy the `/proc` file system of the running processes, as well as kernel state table before you shut down the system, which would cause you to lose this potentially valuable data.

2. Start netcat (**nc**) in listening mode on the safe system (for example, on a laptop or other device that you are sure has not been compromised) such that it will collect the data dumped from the suspect system.

---

```
safe-system# nc -l -p 9000 > /largedisk/memory-dump -w 999
```

---

2. Copy system memory state using **memdump(1)**.

---

```
suspect-system# memdump | nc safe-system 9000
```

---

2. Copy the entire **/proc** file system.

---

```
safe-system# nc -l -p 9001 > /largedisk/proc -w 999  
suspect-system# /cdrom/bin/cp -rp /proc | nc safe-system 9001
```

---

2. Snoop anything coming from the system (see WiFi notes above, and assumes no encrypted traffic). Paranthetical text doesn't seem complete. Please clarify.

Run snoop (or **tcpdump**) on the safe-system, not on the suspect system, to collect any outbound activity originating from the suspect system.

2. After allowing **snoop(1)** or **tcpdump(1)** to run for a period of time, gathering any data possible, finishing the memory dump (**memdump**), and getting a copy of the **/proc** file system, you are ready to put the suspect system into **stop** or **bootprompt** mode.

Perform this system stop on the suspect system using the Stop-A or L1-A key sequence.

---

```
Stop -A
```

---

1 Boot from the forensics CD or from the system install disk (assuming that **dd** and netcat [**nc**] are available on system install disk).

This is done so that you can start working from a known good mini-root kernel and run off of CD for the rest of the steps..

---

```
boot cdrom -S
```

---

2. Optionally, if the CD image supports booting into a windowing system or allows setting up Ethernet hardware and IP addresses to the same Class C network as the safe system, that can be used to help save some steps.

3. At this point, you should be running from the CD on the suspect system and should have plumbed the Ethernet hardware and set the suspect system's IP address

to the same network as the safe system is running. (192.168.1.0 in the example).

If not, run `ifconfig` on the suspect system (running from the CD) to enable the Ethernet hardware and to turn on the network.

---

```
ifconfig hme0 plumb  
ifconfig hme0 192.168.1.10 netmask 255.255.255.0 up
```

---

We are now ready to copy the whole suspect system's disk.

### 2. Netcat listening on safe system.(Please make this a complete sentence.)

---

```
safe-system# nc -l -p 9002 > /largedisk/disk-image -w 999
```

---

---

```
suspect-system# /cdrom/sbin/dd bs=1024 if=/dev/rdisk/c0t0d0s0 | /cdrom/sbin/nc  
192.168.1.11 9002
```

---

2. Repeat the preceding step as needed for all other disks/partitions.

3. Make a second copy of the disk. A simple `/cdrom/forensic/bin/cp image1 image2` will suffice.(Is edit okay?)

All forensic work will be done on the second copy.

## Validate Copies

The purpose of the validation process is to ensure that a confirmed legitimate copy of the system has been made. Use an MD5 checksum for this purpose. You want to ensure that copies of the system exactly match the original suspect system's state. It is especially important that a baseline be created in the event that the copy of the system will be used as forensic evidence in a prosecution.

Use CDROM statically compiled binaries to validate the new copy:

### 1 Create an MD5 checksum on copy 1.

---

```
safe-system# /cdrom/bin/md5-static suspect-drive-root-partiton >suspect-drive-  
root.md5
```

---

1 Create an MD5 checksum on the second copy and compare it to the first copy to ensure they have the same MD5 signature.

This ensures that the two copies are exact copies of the suspect system

## 2. Store the original copy of the system.

Once the copies have been validated, the original copy of the system must be securely stored. This is to ensure that a copy of the system is preserved for future use in the event that prosecution is desired.

## 2. Perform an MD5 checksum of the objects (running binaries from the suspect system).

The following MD5 checksum of the objects (running binaries from the suspect system) are collected, as the MD5 signature can help us later on to identify if the process running was a Solaris binary as shipped by Sun. This helps us eliminate processes that were supposed to be running by comparing them to the Solaris Fingerprint Database.

Note that md5 on Linux has a file size limit of 2 gigabytes, so this might only work on the Solaris OS.

---

```
/cdrom/md5 /largedisk/proc/object/* >/tmp/proc-md5-sigs
```

---

## Prepare for an Analysis

This section is only a short primer and does not describe the full computer forensics process. Entire books have been and are being written about all of the possible steps and procedures involved in this process. At this point, we begin actually looking at the data collected in the preceding steps. The following steps can be done by a third party or at another physical location.

### 1 On a Solaris system, use **lofiadm** to create an association.

---

```
safe-system# lofiadm -a /someplace-with-lots-of-space/suspect- drive-root-  
partition-copy2  
  
/dev/lofi/1
```

---

In the preceding example, you can see that **lofiadm** responded with `/dev/lofi/1`. This is the newly created association between the `suspect-drive-root-partition-copy2` file and a loopback mount point.

### 1 We can now mount READ-ONLY the **dd** image file we called **suspect-drive-root-partition-copy2** in read-only mode using the loopback file association **lofiadm** created for us.

---

```
# mount -o ro /dev/lofi/1 /mnt
```

---

Notice the `-o ro`, or read-only mount.

## 2. Repeat the preceding steps for other disk slices, as needed.

(Optional) Forward a copy of the image to the Sun PS computer forensics lab or other 3rd party which will do the work for you.

At this point, once the `suspect-drive-root-partition` file and the `suspect-drive-root-partition-copy2` file exist as local files, you need to look at the image. The `file(1M)` command says its "data" so we need to be able to mount the "data" file as a UNIX file system to examine it.

Remember, when you are done with your investigation, use the `-d` option to `lofiadm` to remove the association. This is just a reminder. Do not type the following commands at this time.

---

```
# umount /mnt  
# lofiadm -d /someplace-with-lots-of-space/suspect-drive-root-partition-copy2
```

---

**1After verifying that you can mount the disk image, you can shut down the suspected system, install a new disk, and allow system administration to begin recovering and reintroducing the new system into production.**

Only do this if there is a need by the system administration staff to start the rebuild process. There might be information in system memory that can be extremely useful, so if the suspect system can be left in its isolated state, it should be. Having a root disk mirror is a best practice for corporations.

## Perform the Analysis

The purpose of the analysis step is to enable you to determine whether a compromise has occurred, what exactly was damaged, and to obtain any evidence indicating who the culprits might be, as well as their methods.

Tools will be needed to complete the analysis phase. Fortunately, these tools are free and readily available. The tools used in this example are for the Solaris OS and include The Coroner's Toolkit (TCT), mac-robber, as well as Sleuthkit and Autopsy. References for obtaining these tools are provided at the end of this article.

During this task, you are only working from the second copy with read-only capability so you don't disturb any timestamps or corrupt evidence.

1. Start using the md5 checksum and the mactime and create an md5 checksum for every file created in the last 15 days. Using the MD5 checksum and the modified, access, and change (MAC) time, create an MD5 checksum for every file created in the last 15 days.

The MD5 checksums tell you pretty quickly if the intruder left a smoking gun to examine because it allows you to check the MD5 signatures against the Solaris Fingerprint Database.

---

```
safe-system# find /mnt -mtime -15 -exec md5 {} \ ; >mtime-minus15- md5
```

---

1 Grab the MAC timestamps and make MD5 signatures of all the binaries.

---

```
safe-system# mac-robber /mnt1 >/usr/tmp/body.mac  
safesystem# mactime -b /usr/tmp/body.mac 03/25/2003 >timeline.03-25-2003
```

---

The MAC time gives a timeline of what has changed on the system, and the sequence of those changes.

mac-robber is a forensics and incident response tool used to collect the MAC times from allocated files. It recursively reads MAC times of files and directories and prints them in time-machine format to STDOUT. This format is the same that the mac-time tool from TCT reads. mac-robber is based on the grave-robber tool from TCT when using the -m flag, except it does not require Perl.

The loopback mount of the suspect disk has already been mounted using the `lofiadm` command, so you can assume `/mnt` is the mount point in the examples.

1 With the TCT mac-time tool, analyze the `body.mac` created above using `mac-robber`.

---

```
safe-system# mactime -b /usr/tmp/body.mac 10/06/2002 >timeline.10-06-2002  
safe-system# vi timeline.10-06-2002
```

---

Notice the `timeline.$DATE` file shows a chronological ordering of the files. Notice that the `timeline.$DATE` file that is generated when you supply the preceding commands displays the files in chronological order. The quick and dirty way to identify potential attacks is to look for anything modified beginning on the date/time that the break-in was detected by the system administration staff, and try to work backwards from there to see if you can figure out when the initial system penetration occurred. This might be important to determine which back up tapes can be safely used to restore the system.

This technique does not always work; intrusions can go undetected for months before

a user or administrator notices anything. By starting with 15 days from the event, we are hoping to get lucky. If we don't find the break in during this period of time, we will continue going back in additional increments of 15 days until we find answers or run out of time/money.

The following example is an excerpt of a mac-robber timeline from an actual break- in where the intruder left behind the following files and directories. We can deduce the order by which things were installed based on these files modification (m); access (a); or creation (c); date:

Date	size	mac	perms	owne	group	filename
Sep 17 02 11:31:19	11848	.ac	-r-xr-xr-x	root:	staff	/usr/bin/w
	4692	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/cleaner
	8616	.ac	-r-xr-xr-x	root:	staff	/usr/bin/sun3x
	534	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/defines
	512	.ac	drwxr-xr-x	root:	staff	/dev/prom
	17440	.ac	-r-xr-xr-x	root:	staff	/usr/bin/mc68000
	8000	.ac	-r-xr-xr-x	root:	staff	/usr/bin/mc68010
	21424	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/sn2
	26372	.ac	-r-xr-xr-x	root:	staff	/usr/bin/mc68020
	18584	.ac	-r-xr-xr-x	root:	staff	/usr/bin/mc68030
	136288	.ac	-rwxr-xr-x	root:	staff	/usr/bin/wget
	8332	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/pg
	52272	.ac	-r-xr-xr-x	root:	staff	/usr/bin/mc68040
	512	.ac	drwxr-xr-x	root:	staff	/dev
	1819109	.ac	-rw-r--r--	root:	staff	/usr/lib/vold/nsdap/sn.1
	8024	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/utime
	1820577	.ac	-rw-r--r--	root:	staff	/dev/prom/sn.1
	804	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/basepatch
	96252	.ac	-r-xr-xr-x	root:	staff	/usr/bin/sun2
	19452	.ac	-r-xr-xr-x	root:	staff	/usr/bin/sun3
	558868	.ac	-rw-r--r--	root:	staff	/usr/bin/ssh
	4388	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/patcher
	4868	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/crypt
	335	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/sniffload
	4817	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/findkit
	512	.ac	drwxr-xr-x	root:	staff	/usr/bin
	174	.ac	-rwxr-xr-x	root:	staff	/usr/lib/vold/nsdap/README
Sep 17 02 11:31:20	512	.ac	drwxr-xr-x	root:	staff	/usr/lib/vold
	512	.ac	drwxr-xr-x	root:	staff	/usr/lib
	512	.ac	drwxr-xr-x	root:	staff	/usr

⑩ The preceding example is only a small portion of the time slice. Expect to sift through much more data than is shown in this short example.

The obvious fact that these files were accessed (a) and created (c) tells us plenty and gives us places to look at more closely. The first questions you might ask in this case are, "Why would mc6800 a Motorola (sun3) system binary be accessed on a Solaris 2.6 SPARC architecture system, and why would it even still be there?" Most likely, the attacker moved aside old or unpatched binaries to hide some tools. Reading /usr/lib/vold/nsdap/patcher, a script left behind by the intruder,

verifies that this is what happened.

Notice that in this example, the sequence from the oldest directory to the newest, `/usr/lib/vold/nsdap` is created first, then `/usr/lib/vold/nsdap/README` is created, then `/usr/bin` is accessed, `/findkit` is created and run, and then `/sniffload`, `/crypt`, `/patcher`, and `/sshd` are installed, according to the timeline.

Then the following are modified (trojaned); `/usr/bin/sun3`, `/usr/bin/sun2`, then `/usr/lib/vold/nsdap/basepatch` is run, and `/dev/prom/sn.l` (sniffer log) is created. And so on. Then, `wget` is used to get the latest Sun patch bundle after the original unpatched versions are placed in Trojan hiding places, and then the `cleaner` script is run.

You might ask why the attacker did this. By performing these actions, the attacker causes the system to now pass a `patchdiag` or patch checker audits, appearing to be up-to-date on patches. This can also ensure that other hackers can't steal what the attacker has already taken. (Sun doesn't currently have MD5 signatures in patches or in `patchadd`.) The system in this state will have down-reved binaries that are exploitable or in some cases just plain Trojans that hide the attacker's presence, but will appear to be up-to-date in patching. This also prevents a system administrator from reinstalling a system patch that would destroy the attackers backdoors because the system (`patchadd`) will not install a patch that it believes has already been applied.

`Patchdiag` as an auditing tool in this instance would have been totally fooled. The `sidekick.sh` script that checks MD5's of `suid` might have caught this if the auditor had actually run the MD5 outputs against the Solaris Fingerprint Database.

This clever trick is no match for the Solaris Fingerprint Database, but the example demonstrates why auditors can no longer rely on `patchdiag` or `patchadd` when ensuring systems are up to date.

**2. Returning to the analysis tasks we started earlier, let's get the MD5 checksum of everything changed in the past 15 days because we already know many files were modified in that time period.**

We can use this data to determine whether the binaries from our image were shipped by Sun, and from which OS revision they came from. Or, more interestingly, we can determine whether they were NOT Sun-shipped binaries, but are, instead, backdoors left by the intruder to hide their presence.

---

```
safe-system# find /mnt -mtime -15 -exec md5 {} \; >mtime-minus15-md5
```

---

**Analyze the MAC times in the timeline.xDATE file.**

At this point we simply use an editor (vi) on `timeline.01-25-2003` we created. We are looking for files that have been introduced into the system recently. This will show a timeline of events.

By looking at the MAC times we get a picture of when the system recorded the changes.

**Brief Introduction to the Solaris Fingerprint Database, and how it can be used in forensic investigations.**

A full description of the Solaris Fingerprint Database can be found here; <http://sunsolve.sun.com/pub-cgi/show.pl?target=content/content7>. For our purposes, Sun has taken a MD5 cryptographic checksum of every Solaris binary that Sun has ever shipped and put these MD5 checksums into an online database that can be queried by anyone wishing to discover from which OS or patch a Solaris binary came from. We can use this to validate binaries.

`sfpC.pl` is a perl script used to easily query the Solaris Fingerprint Database. We can use it against our collected MD5 signatures to see if our binaries are from Sun or not.

<http://www.sun.com/solutions/blueprints/tools/>

---

```
safe-system# sfpC.pl mtime- minus15- md5
```

---

⑩ The above only looks at the past 15 days which is what we collected earlier.

Earlier we also collected the MD5 signatures of all of the /proc/objects/\*. We can run these against the Solaris Fingerprint Database using sfpC.pl as well to find out what binary the a.out in /proc refers to. Or conversely what binary that was running was not a Sun-shipped binary.

⑩ False positives can result if the system was running any 3rd party software, as these MD5 signatures will not be reflected in the Solaris Fingerprint Database. So do not jump to any conclusions if a file shows up as not having a match in the database. These only indicate binaries that may need further investigation.

---

```
safe-system# sfpC.pl /tmp/proc-md5-sigs
```

---

This will show if the binaries were shipped by Sun and from which OS release.

Things from /proc/objects/\* should still match if they were Sun-shipped binaries, so the a.out that matches Sun-shipped sendmail (as an example) may be perfectly all right if it was running. Mostly we are looking for binaries that have no exact match or have matches from an older version of Solaris that at one time had a security vulnerability.

Make sure and look for older (Solaris 2.6) binaries on newer systems (Solaris 8) Any binaries that do not match may be Trojans Or are not Sun binaries\*

---

```
safe-system# sfpC.pl mtime-minus15-md5
```

---

Example of the Solaris Fingerprint Database's output returned by sfpC.pl:

---

```
b1a9e23c8cb35e34d82e043164c7954e - (/mnt/usr/bin/w) - 1 match(es)
```

```
canonical-path: /usr/bin/uptime  
package: SUN
```

```
-----  
b08a8c2486d048bcd63e79ea1092722a - (/mnt/usr/bin/m68k) - 1 match(es)
```

```
canonical-path: /usr/bin/su  
package: SUNwcsu  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC
```

```
/mnt/usr/bin/m68k is really the original /usr/bin/su
```

```
-----  
e643207218360aa62d57ee078ddce916 - (/mnt/usr/bin/sshd) - 0 match(es)
```

```
Not found in this database.
```

```
Not a Sun shipped binary; and added to the system during the intrusion
```

```
-----  
e0686aeb9f46ff8a2491a09bf2879055 - (/mnt/usr/bin/sun2) - 1 match(es)
```

---

---

canonical-path: /usr/bin/passwd  
package: SUNWcsu  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC  
patch: 106271-08

/mnt/usr/bin/sun2 = original /usr/bin/passwd

-----  
0236041c43c1c7c96cb724f0e41ad5a0 - (/mnt/usr/bin/sun3) - 1 match(es)

canonical-path: /usr/sbin/ping

-----  
4cd4af91f1c1ea0453f53a8clb45c546 - (/mnt/usr/bin/wget) - 0 match(es)

Not found in this database.  
Not a Sun shipped binary; and added to the system during the intrusion

---

---

-----  
30e2cbc4f807920d7ac008db86b64baf - (/mnt/usr/bin/sun3x) - 1 match(es)

canonical-path: /usr/bin/strings  
package: SUNWtoo  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC

/usr/bin/sun3x = original Solaris /usr/bin/strings

-----  
leabd3dbc0746c8a4b5467f99a4f8823 - (/mnt/usr/bin/mc68000) - 1 match(es)

canonical-path: /usr/bin/ls  
package: SUNWcsu  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC

/mnt/usr/bin/mc68000 = original Solaris /usr/bin/ls

-----  
e1693a0caf8551857bc1ae6e2f9a0df3 - (/mnt/usr/bin/mc68010) - 1 match(es)

canonical-path: /usr/bin/du  
package: SUNWcsu  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC

/mnt/usr/bin/mc68010 = original Solaris /usr/bin/du

-----  
162134f40dbc47eb5ed113592bfed10 - (/mnt/usr/bin/mc68020) - 1 match(es)

canonical-path: /usr/bin/ps  
package: SUNWcsu  
version: 11.6.0,REV=1997.07.15.21.46  
architecture: sparc  
source: Solaris 2.6/SPARC

/mnt/usr/bin/mc68020 = /usr/bin/ps

---

Okay by examining the output from sfpC.pl 33.(i.e., the output from the Solaris Fingerprint Database) we can now see those that “match” which are most likely legitimate processes and those that have “zero (0)” or no matches in the Solaris Fingerprint Database. The non-matching ones in our example are ones either started or left behind by our suspected intruder.

---

```
-----
ac9ef190901c01a2c88674c4af38f063 - (/mnt/usr/bin/mc68030) - 1 match(es)

canonical-path: /usr/bin/find
package: SUNWcsu
version: 11.6.0,REV=1997.07.15.21.46
architecture: sparc
source: Solaris 2.6/SPARC

/mnt/usr/bin/mc68030 = /usr/bin/find
-----
6249c10975ac50b8fffe408342191e5b - (/mnt/usr/bin/mc68040) - 1 match(es)

canonical-path: /usr/bin/netstat
package: SUNWcsu
version: 11.6.0,REV=1997.07.15.21.46
architecture: sparc
source: Solaris 2.6/SPARC

/mnt/usr/bin/mc68040 = /usr/bin/netstat
-----
130a263319a918bd3e2bbf5cc5b2c1fa - (/mnt/usr/lib/vold/nsdap/pg) - 0 match(es)
Not found in this database.
-----
cd63323c0eb2c25da98641ad701ee5a7 - (/mnt/usr/lib/vold/nsdap/sn2) - 0 match(es)
Not found in this database.
-----
60811226259b44dc48bf63503f681d6b - (/mnt/usr/lib/vold/nsdap/sn.1) -0 match(es)
Not found in this database.
-----
```

## 2. Other Solaris commands that can assist in computer forensic investigations.

The following examples show how trojan binaries may differ from Sun-shipped commands.

```
⌘ /usr/ccs/bin/what(1)
⌘ /bin/strings(1)
```

Example use of system command what(1)

---

```
root# /usr/ccs/bin/what /bin/login
```

---

Output from /bin/login from a Sun shipped (good) binary

```
/bin/login:
```

```
Sun OS 5.8 Generic 111085-02 November 2001
```

---

```
root# /usr/ccs/bin/what /mnt1/bin/login
```

---

Output from our suspect systems /bin/login binary

```
/mnt1/bin/login:
```

```
Copyright (c) 1980, 1987, 1988 The Regents of the University of California.
```

```
login.c 5.32.1.1 (Berkeley) 1/28/89
```

```
Sun OS 5.5 Generic November 1995
```

Notice that we were not only to validate using MD5 that this was not a Sun-shipped binary but were able (using what(1) ) to see that this binary was actually created from source code from a BSD system.

We perform the same trick against our other suspect system binaries and obtain similar results

---

```
root# what /mnt1/bin/lS
```

---

Output of what(1) on the suspecte "ls" binary

```
ls:
```

```
Sun OS 5.6 Generic August 1997
```

```
root# what /bin/lS
```

/bin/ls:

Sun OS 5.8 Generic February 2000

Notice in this instance the intruder installed a Solaris 5.6 (Sun OS 5.6) "ls" binary in place of the Solaris 8 (Sun OS 5.8) system binary as shipped by Sun in the Solaris 8 OS.

We continue with one more example of using what(1)

---

```
root# what /mnt1/bin/passwd
```

---

Output from our suspect system /bin/password binary

/mnt1/bin/passwd:

Sun OS 5.6 Generic August 1997

---

```
root# what /bin/passwd
```

---

Output from a known good Solaris binary of /bin/passwd

/bin/passwd:

Sun OS 5.8 Generic 111659-05 December 2001

Next we use the strings(1) command to examine some of our suspect binaries

---

```
root# strings -a /mnt1/bin/login | more
```

---

Output using strings(1) of our suspect binary of /bin/login

---

```
/var/spool/lp/.lpr  
[file]  
find  
file_filters  
[ps]  
ps_filters  
[netstat]  
netstat  
net_filters
```

---

We try using strings(1) against the suspect /bin/passwd command as well

---

```
Root# strings /mnt1/bin/passwd |more
/usr/lib/libX.a/uconf.inv
[file]
find
file_filters
[ps]
ps_filters
[netstat]
netstat
Net_filters
```

---

Ah Ha. We see a reference to /usr/libX.a/uconf.inv. What is that doing in there?

One last example using strings(1) but this time using the “-a” argument

---

```
Root# strings -a /mnt1/bin/passwd |more
```

---

Output from the “strings -a” command on our suspect passwd binary

lsof\_filters

shell

**/usr/lib/libX.a/bin/passwd**

**bexter**

/bin/sh

Hmm. What in the heck is the word “bexter” doing in /bin/passwd and again we see a reference to /usr/lib/libX.a directory. This is not normal.

## 2. Inode analysis

Unix and Solaris filesystems (UFS) use inodes to allow the file system to keep track of files. Inodes are created sequentially (inode #5 is created before inode #6) We can use this fact to our advantage while doing a computer forensic investigation.

When a file is created, a new inode is assigned to that file. This is what the filesystem uses to keep track of where the file is located on the physical disk media. When a system is freshly installed, files are copied to the filesystem from the installation media (install CDROM) and the inodes created are in sequential order. Thus one would expect that when the /bin directory was created and populated that the inodes for each of the files in /bin would have inodes where the first file created would have a lower inode number and all the files in /bin that were created by the installation would have their inodes in the same sequential range. Inodes numbers are not reused unless the inode table on a system has been exhausted, in common practice the higher the inode number the newer the file creation time. This is a rule of thumb. Your mileage may vary.

We use the “-i” argument to “ls” command to show us the corresponding inode associated with the file.

---

```
safe-system# ls -i /bin/am*
```

---

Condensed output from “ls -i /bin/\*”

<b>inode</b>	<b>file</b>
81609	amicert
81610	amicertify
81611	amidecrypt
81612	amiencryp
81613	amikeystore
81614	amilogin
81615	amilogout
81616	amisign
81617	amiverify

---

Notice that in a system installation that the inode numbers are sequential beginning with 81609 in our example table above, and each binary afterwards is created with an inode +1 as the files were copied onto the filesystem.

A clue for computer forensics is when the inodes are not in sequential order. This would indicate changes were made to the filesystem. Additionally this can also point to where the file was originally created on our filesystem, and where it first appeared

on the system. If we were to look at our suspect files we could deduce not only at what relative order they were created, but also possibly where on the file system they were first created by examining inodes that are around the same number range. Inode analysis; typical Solaris 8 system

---

```
safe-system# ls -li /usr/bin/lo*
81413 -r-xr-xr-x 1 root bin 172768 Jan 5 2000 /usr/bin/localedef*
81414 -r-xr-xr-x 1 root bin 7204 Jan 5 2000 /usr/bin/logger*
81415 -r-s--x--x 1 root bin 29200 Jan 5 2000 /usr/bin/login*
81416 -rwxr-x--- 1 root bin 12924 Jan 5 2000 /usr/bin/logins*
```

---

Notice how all the inodes are in sequential order. This is typical and normal.

Now let's look at the inode layout from our suspect system.

---

```
safe-system# ls -li /mnt1/usr/bin/lo*
81413 -r-xr-xr-x 1 root bin 172768 Jan 5 2000 /usr/bin/localedef*
81414 -r-xr-xr-x 1 root bin 7204 Jan 5 2000 /usr/bin/logger*
769843 -rwsr-xr-x 1 root bin 28008 Mar 18 09:31 login*
81416 -rwxr-x--- 1 root bin 12924 Jan 5 2000 /usr/bin/logins*
```

---

Notice that the inode numbering is out of sequence for /usr/bin/login! This is yet another strong indication that the original /bin/login program was replaced by another version, and the inode number 769843 also potentially tells us where the file was created

From our previous loop back mount of suspect partition:

---

```
safe-system# find /mnt1 -type f -exec ls -li {} >inode-list \;
```

---

We can go looking for an inode number sequence roughly matching the same range.

We find a directory /usr/lib/libX.a that has contents that have similar inode numbers.

We also find other binaries with similar inodes. These become suspects of also being modified or even Trojans.

---

```
769828 -rwsr-xr-x 1 root bin 18844 Mar 18 09:31 ls
769835 -rwsr-xr-x 1 root bin 101744 Mar 18 09:31 passwd
769842 -rwsr-xr-x 1 root bin 9492 Mar 18 09:31 ps
769826 -rwsr-xr-x 1 root bin 17156 Mar 18 09:30 su
```

---

This points us at this new directory "/usr/lib/libX.a" to examine. Also notice that this is a directory and not what we would expect. We would expect that a "libX.a" would be an archive library or a "file" in a broader terminology but instead we have a "directory".

We know at this point with some degree of certainty that it was not an accidental modification by an administrator, and at least a partial list of which files have been replaced. We don't know at this point all files that were examined or potentially modified, but we do know that we are on the right track.

We can now make a decision to do a more in-depth analysis using more sophisticated forensic tools. We can also choose to go back and re-install the production machine and get it into production.

One last tip. Internet search engines have become a wealth of information as well as pointers to others who might have suffered the same break-in and might have possibly already done some of the same analysis you are currently doing. Why not take advantage of this wealth of information? Going to an Internet-connected Web browser, and going to our favorite search engine and searching on things we have found that are suspicious in our binaries gives us information about what toolkit was used, and links to analysis that has already been done by others.

<http://www.google.com>

Search for ' /usr/lib/libX.a bexter rootkit/

## Perform Recovery

This article describes the steps for performing a forensic analysis. The recovery that should be performed after that analysis is really the standard process that one would normally follow, and as such is beyond the scope of this article. In general and to ensure that your forensic analysis was not a wasted effort, it is suggested that when reinstalling your system, do not forget to patch it, harden it, minimize it, and apply a defense in depth strategy. Please refer to the Sun BluePrints OnLine Web site for related details. <http://www.sun.com/security/blueprints>.