



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Forensic UNIX Initial Response Script and CDROM – Collect the evidence that will be lost by disconnection or shutdown

Summary

The first few minutes of an information security incident are often the most critical. The initial response often dictates what actions are available for the company down the road. Unfortunately, without preparation, you are likely to limit your options and lose evidence you will wish you had saved. Often the first actions management or the situation suggest will destroy evidence that can never be recovered again. This paper aims to provide you with the background and tools to prepare an initial response to the incident that will give you the most options as the investigation continues. The script was written for and using Red Hat Linux 8.0. However, many of the same commands will work on other systems and the philosophy and procedures will certainly apply to every system. The idea behind this paper is to be able to quickly collect the information that will be lost if management orders the compromised system to be shutdown or disconnected before a proper forensic evaluation is completed. With that data easily and quickly captured, you are free to follow the management decision and proceed with a more detailed analysis later if appropriate.

Real World Example

Let us start with an example from the real world. This example is not intended to be a model or checklist of incident handling. Indeed, this is NOT the way that I would do it again. It is merely intended to provide you with an example situation that the script and CDROM in this paper would be well-suited for. Indeed, it was the situation that prompted the thinking that led to this paper.

I was working on very important system administrator stuff at my office in San Diego when I got the phone call. It was a fairly tech-savvy user at the branch office in Santa Clara. He reported that a partner company was having trouble connecting to the Santa Clara Office FTP server through the Internet. I began the usual troubleshooting. I asked if the FTP server was on. He said it was. I then pinged the FTP server and it was responding, so I tried to connect with a FTP client and was told that the maximum number of users had already connected. That is when I started to feel funny. This FTP server was only used for a handful partner companies. It was rare for even two users to be connected at the same time.

So I logged on to the server with telnet (it was an old installation and did not have SSH installed). I had just gotten the shell prompt when I was suddenly disconnected. A second attempt to login resulted in the same

disconnection. I had the user log on to the console and he was successful, so we could at least get a prompt that way. While he was doing that, he reported that there was an excessive amount of disk activity. Especially unusual when it seemed no authorized users were connected to it.

I reviewed the background and details about the server. Often the background of a machine can give clues as to what might be possible. It had been installed by one of the users (not the IT department) a few years ago. It was running a default installation of Linux (Red Hat 5) that was several versions out of date. It had never been patched or updated. It was exposed to the Internet and was not protected by any firewall protection or anti-virus. It had been scheduled for replacement for a long time and was not on a list of IT department maintained servers.

I now suspected that the machine had been compromised. The version of Linux that it was running had several well-known vulnerabilities that would allow someone to remotely take full control of the machine. As it was a default installation, had never been patched, and was facing the Internet, it seemed likely that someone scanning the Internet for vulnerable machines had found it, compromised it, and was now using it for who knows what. The other possibility was that there was a system problem that was eating up system resources (causing the hard drive activity), and preventing remote connections. I thought compromise was more likely. So what was the system doing? Was it being used to attack other systems? Who was connected to it?

Now came the time to call my boss. I reported what was happening and what I saw as our options. I told him Option 1 would be to try to investigate what was happening, find out how the machine was compromised, and what it was doing right now. Option 2 would be to take the machine down, and wipe the disk clean. Then install a secure operating system, protect it with a firewall, and keep it up to date. While we were speaking, the FTP server kept churning away with multiple unknown people connected to it. I knew that if we powered down the machine, we would lose important evidence, but the longer it was left on, the more damage could be caused. What would we do?

Many Questions in the First 5 Minutes

The clock starts with the first sign of trouble. Something has indicated that a system compromise may have occurred. The system logs maybe showed tell-tail signs; mysterious files were appearing; the machine was just behaving strangely; people were connecting to the machine that had no business being connected.. You, as the first IT department representative on scene, have several questions already.

Is it real? This could be a false alarm. Authorized access or system behavior that only looks like an intrusion? It could be a malfunction or bug in the system that is causing the strange behavior. Is it unauthorized but harmless? Is an employee storing a few music files temporarily before

burning them to a CDRW?

What is happening? How did they gain access to the system? Where is the security breach? On this machine or another one? What exactly are they doing? Are they deleting files? Sharing files? Launching attacks on other systems?

Is harm being done to the machine as we stand here and think? Are company secrets leaving by the second while we ask ourselves these questions? Or are we looking at the echoes of an event that is now distant history. Is our machine actively involved in attacking other hosts?

Who is doing it? Company employee? Ex-employee? Contractor? Stranger from the outside? A lone hacker or is a group at work here? Not a human at all but some Internet worm or virus?

Why are they doing it? Simple cyber joy riding? Using the company's machine to attack other hosts? Setting up a base from which to communicate and share files with other hackers? Is this a sophisticated attacker supplying trade secrets to your competitor?

What are we going to do about it? Nothing and hope it is not serious? Quickly power down the system? Try to close the security hole? Unplug the network cable? Notify upper management? Notify customers? Reinstall everything? Try to figure out who did this? Or just stop the attack? Try to criminally prosecute the offenders? Many people will need to be involved in the decision. Management will have to decide if systems can be taken down for analysis or if it is critical that systems remain up. Legal counsel will have to decide if any laws or agreements require notification of customers or partners. IS staff will have their recommendations on how to close the security gap and recover the system.

All of these discussions and decisions will take more than a few minutes to resolve and yet you must do something now. Often, there is great pressure to hit one of the following panic buttons. Sometimes the pressure comes from management wishing to stop the loss or the liability.

Panic Button Number 1: Pull the power plug on the system.

Panic Button Number 2: Pull the network cable out of the back.

Either certainly may put a stop to whatever is happening on the system. No more data will leave the system. File deletion may stop. If the system is attacking other hosts, that will be stopped. It will also:

- 1.Alert the intruder that someone noticed.
- 2.Destroy evidence that may never be recovered such as:
 - a) Active network connections.
 - b) Processes that are running at the time and what files they have open.
 - c) File modification/access dates.

- d) Some files themselves.
- e) Anything in the swap or /proc space.

Panic Button Number 3: Start entering commands on the system. Often this temptation is hard to resist. You have a natural urge to confirm your suspicions before you invest any more time on it. You would hate to sound the alarm, or spend more time away from the six other things you are working on, if it is really nothing. However, entering commands on the system without a plan may make it harder to determine what is going on. Also, as there is some information that can only be retrieved once, you may not be in a position to record the results of your commands in a manner that will allow court or administrative action later. You do not want to do anything that will limit your options later.

Probably the biggest reason you do not want to just start entering commands on a possibly compromised system is that you can not trust the results the commands give you. One common tactic that UNIX hackers take is to install a root kit as soon as they have compromised the system. A root kit replaces system utilities like **ps**, or **ls** with trojanized versions. These versions will give exactly the same results as the real utilities, with the exception that they will hide the hacker's activities. For example, a trojanized **netstat** may show all network connections EXCEPT those coming from the hacker's machine. A trojanized **ps** may show every process EXCEPT the hacker's keyboard logger. The trojanized **ls** may show every file EXCEPT the hacker's password log. Thus, everything will look normal with no suspicious activity.

There are certainly more options besides these 3 panic buttons that you could take during an intrusion. You might consider blocking the intruders' IP addresses at your firewall, assuming you could quickly determine them. This has it's own implications as you may be blocking legitimate addresses if there's spoofing involved. There also could be other logs you could check; remote syslog servers, firewall logs, intrusion detection logs, etc. There are many other options and a full discussion of incident handling could (and does) fill volumes. This script is for the situation where the administrator on scene does not know anything more than the system is likely compromised and management wants it shutdown or unplugged to stop the damage. In this case, there is not much time for analysis.

When thinking about what actions you might take initially, it pays to think about what your ultimate goal might be. In their presentation on "Investigating Computer Security" at Interop2002, Kevin Mandia and Chris Prosise from Foundstone, Inc. offer four reasonable possibilities for response:

1. **Ignore the incident.**
2. **Defend against further attacks by implementing technical remedies.** This would include things like applying a software fix to close the security hole that allowed the compromise. It might include reinstalling the operating system to make sure the attackers have left no back doors. Perhaps something as

simple as adding a firewall rule might stop the attacks. The point here is that no effort is made to closely determine what damage was done by whom and when. Just fix the problem and get back to work. This is the response I have most often seen in the real world.

3. **Defend against further attacks by implementing technical remedies AND collecting evidence for criminal arrest or civil remedies.** This decision might be based on the amount of damage done, or legal requirements to notify non-company personnel.
4. **Perform surveillance and counterintelligence data gathering.** In this case, you would not close the security hole or try to stop the attackers. You would setup surveillance methods and tools in order to gather as much information as possible about who the attackers are and what they are doing. The ultimate goal of which would be to prosecute or take other remedies against them. The downside of course is that the damage may continue while you watch.

One problem is that you may not be able to determine in those first few minutes how much damage has been done or how much information has been lost. For instance, you may not discover until it is accessed days later that critical data has been deleted or modified; data that you did not think was compromised at first. Or you may not realize that your customers' account numbers were compromised. So it may not be apparent at the beginning that an investigation is warranted.

Mandia and Proise go on to say that whatever response or actions you take in those first five minutes must:

1. **Support responses 2, 3, and 4.**
2. **Be forensically sound.**
3. **Be simple and efficient.**
4. **Provide an accurate snapshot for decision makers.**
5. **Support civil, administrative, or criminal action.**

There is a lot to do and think about in those first few minutes. Unfortunately, there is a lot of heat/pressure from the situation. As time passes, more damage may be done to your company, more damage may be done to the evidence, etc. The questions listed so far are going to take some time to work through. Unfortunately, because of the potential of ongoing damage or harm, there is often a great deal of pressure to hit one of the panic buttons.

Ideally, the company has developed an Incident Response (IR) Plan. Putting this plan together would force the company to think through the questions listed in this section in advance. Once the plan is in place, the company already knows who needs to be notified, who is responsible for what actions, where the company's priorities are, etc. While the development of an IR Plan is not the focus of this paper, it is vital for your company. There are

many resources that can assist you in this process. One is a web-based guide by the Carnegie Mellon Computer Emergency Response Team Coordination Center. The URL is listed in the references.

So what do you do?

The key thing is that you DO NOT want to do anything that would limit your options down the road. In other words, you want your initial response to support any follow-on action that management may decide to take. Fortunately, if you plan on preserving everything to a level that will stand up in a court, then you are ready for any eventuality.

With some advance preparation, you will be able to capture the volatile information that will be lost by disconnecting the network cable or powering down the system. You will also capture it in a manner that will hold up under the scrutiny of legal or administrative action. Once that volatile data and evidence has been captured, you are free to power down the system and decide what action to take next. You will not have closed any options, or destroyed evidence that can never be recovered again. Often, the information gathered in this initial response, will help you determine the extent of the compromise.

First Response Script Strategies

A good solution is to prepare a CDROM in advance of the compromise. This CDROM will contain trusted versions of system utilities. Versions that you know have not been compromised. Also on the CDROM is a script that uses those utilities to gather the most volatile evidence to floppies; the evidence that will be lost when you power down or disconnect the system. At the first sign of compromise, you run the script on the CDROM. Then you have much more freedom to shutdown the system and take the time to decide what to do next. In addition, the script can give you the initial assessment of how serious the compromise might be.

The initial response CDROM will contain the following:

1. Shared libraries from a trusted source (if you do not use statically linked tools). More about this later.
2. BASH shell from a trusted source.
3. System tools from a trusted source.
4. Initial response script using those tools.
5. MD5 checksums of the script. MD5 is an algorithm that can parse any file, and generate an alphanumeric checksum or "signature" of that file. A file that is different by even one character will have a different signature.

The MD5 algorithm is described in Internet Engineering Task Force Request for Comments (IETF RFC) 1321. There it says in the executive summary:

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest.

This allows you to prove at a later date (in court for example) that the script you ran is the one you say you ran. It also helps you prove that your evidence has not been tampered with by comparing the MD5 checksum, or signature, generated when it was collected with the MD5 checksum generated in court. When they match, you can be confident that the files are the same.

Any investigation also needs the following:

1. Floppies to hold the data generated by the script.
2. Note pad for taking notes.
3. Lockable box to hold notes/floppy as evidence.
4. Results of test runs of script against key systems before compromise to provide baseline.
5. A printout of the script and a printout of the MD5 checksum.

Why a CDROM? There are lots of options for the location of the script when you run it. You could put the script on a floppy or on a file server and run it from the compromised machine over the network. So why use a CDROM? Because it is:

1. A read-only format that will allow you to prove later what script/commands were run.
2. Portable. Most modern systems have CDROM's.
3. Big enough to hold all shared libraries to allow you to establish a trusted shell and run trusted programs.

Why a script? Once you have the CDROM with your tools, you could just use it to start entering commands to figure out what is going on. So why write a script in advance? A script provides you with:

1. Automatic documentation. You do not have to remember what commands you entered or in what order. MD5 and read/only media can prove you ran what you say you ran.
2. Fast results. The script can execute and record the evidence much faster than you could by typing. So fast that perhaps the most vigilant intruder will not be aware that someone is looking until it is too late.
3. An easy-to-baseline toolkit. You can easily run the exact set of commands multiple times per system to establish baselines from which to compare.
4. A testable toolkit. You can thoroughly test the script in advance. Once the bugs are worked out, you do not have to worry about making a mistake in the heat of battle.

5. An initial response that does not require forensic expertise. Since the script can be run with a single command, any end user can run it. You can quickly gather the volatile evidence, then shutdown the system without being on site (as long as each site has a copy of the CD).

Why floppies? Why not just store the evidence on the system itself? Because:

1. You want to avoid writing to the victim system. Anything you write affects the modification times and could overwrite important data.
2. They are portable. You can easily move them from system to system.
3. They are easily controlled. You can lock them up, label them, and even put the write-protect tab in effect.
4. You could burn the data to a CD when you are done for added protection.

While this paper is not about writing company policies, it is important to mention the minimum requirements for this strategy to be effective. At a minimum, you must set the following policies regarding suspected compromise:

1. No more commands are entered on the system.
2. No more logins/connections to the system.
3. Only IT department personnel unplug/shutdown the network or system.
4. Before system is unplugged or shut down, the initial response script must be run.
5. Discussions with appropriate technical and management staff will be held to discuss the next courses of action.

The key part of this policy is that you must get the word out to the users. Not just the usual memo in the mailboxes that no one reads, but actually understood to the point that six months from now, when the suspected compromise occurs, the user will remember not to shut down the machine. One user or manager on scene that unplugs the network cable or shuts down the system at the first sign of trouble will make all your advance work useless.

Now for the script where the actual work will be performed. You must decide on what tools need to be run. The order of the commands is important. You want to capture the most volatile evidence first, and then move on to the less volatile. In fact, this article only focuses on the evidence so volatile that it will be lost if you power down the system or disconnect the network. It is also intended to help you get a better idea if there is an actual compromise in progress. Who is connected and from where, what programs are running, etc...

This script is not the end of the investigation by any means. Much work needs to be done after this script is run. To do a proper investigation, you would have to account for each program running on the machine, do a scan for backdoors or mysterious files, look for strange account creation, check log files for evidence, etc. However, much of this work could be done after powering down the system or doing a forensic duplication. As it will take a long time to perform this investigation, you can take your time and do it right. We are going to focus on the first few minutes, when you do not have time to look anything up and have to do something fast before management insists that the compromised machine be shutdown.

The importance of a trusted environment.

It was mentioned earlier that one of the reasons you would not want to just start entering commands on a compromised system is that you would not be able to trust the results. If a root kit has been installed, **ps** may not list all processes; it may be hiding the hacker's key logger. So all we need to do is copy a trusted version of **ps** to our CDROM and we are done, right? Unfortunately, it is not that easy. In order to have a trusted program, you must have a trusted shell from which to run it, the trusted program itself, and trusted versions of any shared libraries it is using.

Common system calls are often made by several utilities in the same ways. A common tactic programmers take when writing code, even system utilities, is putting code common to more than one program in a shared library. Then in the original program, all you need is a link to the shared code. This brings several advantages. The first is that your programs and utilities can be much smaller than if you included that common code in each of them. Another advantage is that you can make changes to the shared libraries to keep up with changes and improvements in the operating system without recompiling the programs that use them.

In reality, the link in the original program to the shared code is really a link to a "linker" program. In Red Hat Linux 8.0, the linker program is called **ld-linux.so.2**. This linker program checks versions and other services, goes to the actual shared library and returns the code to the program that asked for it. The path is hard-coded into programs that use it. While you will be able to bring all the shared libraries that might be used on your CDROM, the one thing you will not be able to specify to run from your trusted CDROM is **ld-linux.so.2**. However, you could run a MD5 checksum against the **ld-linux.so.2** file on the system and compare it to a trusted version to ensure the file has not been tampered with.

The other solution to this is to use statically compiled utilities that do not use any shared code. That way you can trust the whole program. Hal Pomeranz has posted a page on statically linking programs you compile under Solaris. This is helpful as Solaris does not include a compiler by default. His page can be found here:

<http://www.deer-run.com/~hal/sol-static.txt>

In some cases, it will not be possible to use statically compiled utilities. Either the source code is not available, or the system does not have a compiler installed. So how does one find out which, if any, shared libraries a program uses? The command **ldd** lists which shared libraries are required for the program specified. For instance, the command, "**ldd /bin/ls**" will show you what shared libraries are required by **/bin/ls**. In addition, it will show you the path to the specific shared library that will actually get used if the command is executed. This will be useful to demonstrate that the trusted shared libraries will be used rather than those on the compromised machine.

So for each utility that your first response script is going to use, you must have a trusted copy of any shared library that it uses. Fortunately, all 15+ tools in this script only call eight shared libraries between them. It turns out that most system utilities share a small set of shared libraries. It is not hard to copy those eight shared libraries to the response CDROM.

In the interest of exploring every possible area of exploitation, it should be noted that the script is recording the evidence to a floppy attached to the machine. If the machine is compromised and the intruder becomes aware of your evidence collection, it is conceivable that he or she could attempt to modify your evidence there. As the script completes in a relatively short period of time, this risk may be low. One way to help mitigate this risk might be to sign your evidence using PGP in addition to the MD5 checksum. That capability would make a good addition to the script.

Finally, the Script

What follows is a listing of each command (in bold) in the script along with an explanation of why it was included, what evidence it is collecting, and example output of the command (directly after the command and in a different font). To obtain the output, I simply cut/pasted the contents of the *.txt evidence file created by each command. The example output is from the machine used to develop the script and is not a compromised machine. As such, this would be a good example of using the script to test and baseline a system. Keep in mind that this paper is not about forensic analysis. There is not a lot of discussion on how to interpret the data collected by this script. However, there is some discussion in explaining what evidence each command collects and why it is important.

The first commands set the environment variables necessary to ensure that the trusted versions of our shared libraries are used and not any on the compromised system. It also sets the default path to be our CDROM tools instead of system tools to ensure that only the trusted programs are run. The **LD_LIBRARY_PATH** variable tells the linker program where to look for the shared libraries before looking in the default location. This will let us substitute trusted copies of the shared libraries instead of those on the compromised system. After this environment is set, the only code that is run

that is not on our trusted CDROM is the linker program **ld-linux.so.2**. This we can verify has not been modified with a MD5 checksum.

```
#Set environment variables and establish trusted shell
PATH="/mnt/cdrom/bin"
LD_LIBRARY_PATH="/mnt/cdrom/lib"
export PATH
export LD_LIBRARY_PATH
```

These first commands provide no output.

The next line of the script records the date and time that we started.

```
date > /mnt/floppy/startdate.txt
Output:
Tue Apr 8 14:38:56 PDT 2003
```

Next we will record that we are operating in a trusted environment by showing that our evidence floppies are initially empty. Otherwise someone might make the argument that our evidence did not come from the compromised system.

```
ls -al /mnt/floppy > /mnt/floppy/startdirempty.txt
Output:
total 11
drwxr-xr-x 2 root root 4096 Apr 8 14:38 .
drwxr-xr-x 4 root root 4096 Dec 1 08:40 ..
```

Now, in order to help prove that we are running only trusted code, we record on our evidence floppies the environment variables that pertain. In addition, we run the **ldd** command against our own trusted toolset on the CDROM. The results of that command will show that only trusted shared libraries are used.

```
echo $PATH > /mnt/floppy/path.txt
Output:
/mnt/cdrom/bin
echo $LD_LIBRARY_PATH > /mnt/floppy/libpath.txt
Output:
/mnt/cdrom/lib
ldd /mnt/cdrom/bin/* > /mnt/floppy/ldd.txt
Output:
/mnt/cdrom/bin/date:
    libpthread.so.0 => /mnt/cdrom/lib/libpthread.so.0 (0x40013000)
    librt.so.1 => /mnt/cdrom/lib/librt.so.1 (0x40044000)
    libc.so.6 => /mnt/cdrom/lib/libc.so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/echo:
    libc.so.6 => /mnt/cdrom/lib/libc.so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/finger:
    libc.so.6 => /mnt/cdrom/lib/libc.so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/ifconfig:
    libc.so.6 => /mnt/cdrom/lib/libc.so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

```

/mnt/cdrom/bin/last:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/ldd:
    not a dynamic executable
/mnt/cdrom/bin/lddlibc4:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/ld-linux.so.2:
    statically linked
/mnt/cdrom/bin/ls:
    libtermcap.so.2 => /mnt/cdrom/lib/libtermcap.so.2 (0x40013000)
    libacl.so.1 => /mnt/cdrom/lib/libacl.so.1 (0x40018000)
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    libattr.so.1 => /mnt/cdrom/lib/libattr.so.1 (0x4001e000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/lsof:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/md5sum:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/more:
    libtermcap.so.2 => /mnt/cdrom/lib/libtermcap.so.2 (0x40013000)
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/netstat:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/ps:
    libproc.so.2.0.7 => /mnt/cdrom/lib/libproc.so.2.0.7 (0x40013000)
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/rm:
    libacl.so.1 => /mnt/cdrom/lib/libacl.so.1 (0x40013000)
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    libattr.so.1 => /mnt/cdrom/lib/libattr.so.1 (0x4001e000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/top:
    libproc.so.2.0.7 => /mnt/cdrom/lib/libproc.so.2.0.7 (0x40013000)
    libncurses.so.5 => /mnt/cdrom/lib/libncurses.so.5 (0x40020000)
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
/mnt/cdrom/bin/who:
    libc.so.6 => /mnt/cdrom/lib/libc. so.6 (0x42000000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)

```

The output lists the binary (i.e. /mnt/cdrom/bin/who:) followed by a listing of the libraries required (the files with “so” in the file name) and the physical path to the library. This demonstrates that with the exception of ld-linux.so as discussed above, all code that is running resides on the CDROM.

We now record the MD5 checksum of our script. This will allow us to prove later that the commands listed were the same as those actually run.

```

md5sum /mnt/cdrom/script > /mnt/floppy/scriptmd5.txt
Output:
c010087fb7e3aa5444a77f6587bdd6b7  script

```

Now that we have established and demonstrated that we have a trusted shell and are running trusted utilities, we can begin to collect evidence. We begin by collecting the most volatile evidence first. We list information about which processes are running. The **ps** command lists running processes. The “a” option lists all processes, even those owned by other users. The “u” option specifies a user-oriented format. The “x” option lists all processes without a controlling terminal. The “w” option specifies wide output which will not truncate the line if the information gets too long. This combination should show us all the processes that are running.

```
ps -auxww > /mnt/floppy/ps.txt
```

Output:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	1336	480	?	S	12:20	0:04	init
root	2	0.0	0.0	0	0	?	SW	12:20	0:00	[keventd]
root	3	0.0	0.0	0	0	?	SW	12:20	0:00	[kapmd]
root	4	0.0	0.0	0	0	?	SWN	12:20	0:00	[ksoftirqd_CPU0]
root	5	0.0	0.0	0	0	?	SW	12:20	0:00	[kswapd]
root	6	0.0	0.0	0	0	?	SW	12:20	0:00	[bdflush]
root	7	0.0	0.0	0	0	?	SW	12:20	0:00	[kupdated]
root	8	0.0	0.0	0	0	?	SW	12:21	0:00	[mdrecoveryd]
root	12	0.0	0.0	0	0	?	SW	12:21	0:00	[kjournald]
root	68	0.0	0.0	0	0	?	SW	12:21	0:00	[khubd]
root	161	0.0	0.0	0	0	?	SW	12:21	0:00	[kjournald]
root	407	0.0	0.1	1400	536	?	S	12:21	0:00	syslogd -m 0
root	411	0.0	0.1	1336	428	?	S	12:21	0:00	klogd -x
rpc	428	0.0	0.1	1484	532	?	S	12:21	0:00	portmap
rpcuser	447	0.0	0.2	1528	728	?	S	12:21	0:00	rpc.statd
root	503	0.0	0.2	1456	644	?	S	12:21	0:00	/sbin/cardmgr
root	541	0.0	0.1	1328	476	?	S	12:21	0:00	/usr/sbin/apmd -p 10 -w
5 -W -P /etc/sysconfig/apm-scripts/apmscript										
root	630	0.0	0.4	3276	1468	?	S	12:22	0:00	/usr/sbin/sshd
root	677	0.0	0.2	2092	904	?	S	12:22	0:00	xinetd -stayalive -
reuse -pidfile /var/run/xinetd.pid										
lp	691	0.0	0.3	4352	1072	?	S	12:22	0:00	lpd Waiting
root	711	0.0	0.7	5040	2268	?	S	12:22	0:00	sendmail: accepting
connections										
smmsp	721	0.0	0.6	4856	2048	?	S	12:22	0:00	sendmail: Queue
runner@01:00:00 for /var/spool/clientmqueue										
root	731	0.0	0.1	1372	428	?	S	12:22	0:00	gpm -t ps/2 -m
/dev/mouse										
root	740	0.0	0.1	1512	612	?	S	12:22	0:00	crond
xfs	771	0.0	1.0	4532	3244	?	S	12:22	0:00	xfs -droppriv -daemon
daemon	789	0.0	0.1	1368	520	?	S	12:22	0:00	/usr/sbin/atd
root	798	0.0	0.1	1316	404	tty1	S	12:22	0:00	/sbin/mingetty tty1
root	799	0.0	0.1	1316	404	tty2	S	12:22	0:00	/sbin/mingetty tty2
root	800	0.0	0.1	1316	404	tty3	S	12:22	0:00	/sbin/mingetty tty3
root	801	0.0	0.1	1316	404	tty4	S	12:22	0:00	/sbin/mingetty tty4
root	802	0.0	0.1	1316	404	tty5	S	12:22	0:00	/sbin/mingetty tty5
root	803	0.0	0.1	1316	404	tty6	S	12:22	0:00	/sbin/mingetty tty6
root	804	0.0	0.9	12752	2936	?	S	12:22	0:00	/usr/bin/gdm-binary -
nodaemon										
root	849	0.0	1.0	13472	3468	?	S	12:22	0:00	/usr/bin/gdm-binary -
nodaemon										
root	850	18.7	2.8	19960	9256	?	S<	12:22	25:39	/usr/X11R6/bin/X :0 -
auth /var/gdm/:0.Xauth										
root	859	0.0	2.5	16260	8100	?	S	12:22	0:00	/usr/bin/gnome-session
root	917	0.0	0.3	2900	992	?	S	12:22	0:00	/usr/bin/ssh-agent
/etc/X11/xinit/Xclients										
root	928	0.0	1.3	7820	4376	?	S	12:22	0:01	/usr/libexec/gconfd-2 9
root	930	0.0	0.5	2880	1656	?	S	12:22	0:00	esd -terminate -nobeeps
-as 2 -spawnfd 13										
root	939	0.0	0.6	5588	2200	?	S	12:22	0:00	/usr/libexec/bonobo-
activation-server --ac-activate --ior-output-fd=15										
root	941	0.0	1.9	11724	6132	?	S	12:22	0:01	/usr/bin/metacity --sm-
client-id=default1										
root	943	0.0	2.2	16024	7220	?	S	12:22	0:01	gnome-settings-daemon -
-oaf-activate-iiid=OAFIID:GNOME_SettingsDaemon --oaf-ior-fd=12										
root	947	0.0	0.3	2616	1268	?	S	12:22	0:00	fam
root	954	0.0	3.2	18020	10456	?	S	12:22	0:02	gnome-panel --sm-
client-id default2										

```

root      956  0.0  4.4 36232 14236 ?      S    12:22   0:05 nautilus --no-default-
window --sm-client-id default3
root      958  0.3  1.7 14324 5700 ?      S    12:22   0:31 magicdev --sm-client-id
default4
root      961  0.0  1.2 10444 3844 ?      S    12:23   0:00 pam-panel-icon --sm-
client-id default0
root      963  0.6  3.9 21940 12596 ?      S    12:23   0:56 /usr/bin/python
/usr/bin/rhnm-applet-gui --sm-client-id default5
root      964  0.0  0.1  1364  476 ?      S    12:23   0:00
/sbin/pam_timestamp_check -d root
root      972  4.8  3.0 18220 9612 ?      S    12:34   6:01 /usr/bin/gnome-terminal
root      973  2.0  0.4  4212 1536 pts/0    S    12:34   0:04 bash
root     26590  0.0  0.4  4200 1512 pts/1    S    13:33   0:00 bash
root     27908  0.0  0.2  3396  736 pts/1    S    13:33   0:00 man ./grave-robber.1
root     27909  0.0  0.3  3812  964 pts/1    S    13:33   0:00 sh -c (cd
/home/shared/tct/tct-1.11/man/man1/../../ && (echo ".pl 1100i"; /bin/cat
'/home/shared/tct/tct-1.11/man/man1/./grave-robber.1'; echo; echo ".pl \n(nlu+10") |
/usr/bin/gtbl | /usr/bin/nroff -c -mandoc | /usr/bin/less -isr)
root     27910  0.0  0.3  3812 1012 pts/1    S    13:33   0:00 sh -c (cd
/home/shared/tct/tct-1.11/man/man1/../../ && (echo ".pl 1100i"; /bin/cat
'/home/shared/tct/tct-1.11/man/man1/./grave-robber.1'; echo; echo ".pl \n(nlu+10") |
/usr/bin/gtbl | /usr/bin/nroff -c -mandoc | /usr/bin/less -isr)
root     27915  0.0  0.1  1708  616 pts/1    S    13:33   0:00 /usr/bin/less -isr
root     2443  0.0  0.2  2740  772 pts/0    R    14:38   0:00 ps -auxww

```

The “proc” directory tree is a virtual tree that contains information about which processes are running. The Man Page Proc (5) says, “/proc is a pseudo-filesystem which is used as an interface to kernel data structures rather than reading and interpreting /dev/kmem.” A plethora of information about processes can be found in this tree that can not be gleaned from a **ps** listing. Under the /proc directory is a directory for each process with the Process Identification (PID) as the name of the directory. In each of these PID directories are multiple files that describe that process. There is a file called “cmdline” that contains the exact command line used to launch the process. A link called “cwd” is a link to the current working directory of the process. A link called “exe” is a link to the executable file for the process. Other detailed information can be found that will help determine exactly what a process is doing and how it was launched. It can help you determine which processes might have been started by an intruder to modify or monitor system behavior.

A great explanation of the /proc space is in the book “Running LINUX” by Welsh, Dalheimer and Kaufman listed in the references on pages 147-149.

The first of the two commands does a recursive listing of the “proc” tree to show all links and processes. The next command does a **more** on each file to show the contents of each file. These commands (especially the second one) could take a long time to complete and take up a lot of floppy disk space to store the results. You may want to test this command a few times before including it on your CDROM.

```
#ls -alR /proc > /mnt/floppy/ls-proc.txt
```

Output (Truncated to save space):

```

/proc:
total 4
dr-xr-xr-x  69 root    root          0 Apr  8 05:20 .
drwxr-xr-x  21 root    root        4096 Apr  8 14:50 ..
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 1
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 12
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 161
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 2

```

```

dr-xr-xr-x   3 root    root          0 Apr  8 15:59 2461
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 2527
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 2616
/proc/2461:
total 0
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 .
dr-xr-xr-x  70 root    root          0 Apr  8 05:20 ..
-r--r--r--   1 root    root          0 Apr  8 15:59 cmdline
lrwxrwxrwx   1 root    root          0 Apr  8 15:59 cwd -> /home/shared/tct/tct-
1.11/man/man1
-r-----   1 root    root          0 Apr  8 15:59 environ
lrwxrwxrwx   1 root    root          0 Apr  8 15:59 exe -> /usr/bin/gedit
dr-x-----   2 root    root          0 Apr  8 15:59 fd
-r--r--r--   1 root    root          0 Apr  8 15:59 maps
-rw-----   1 root    root          0 Apr  8 15:59 mem
-r--r--r--   1 root    root          0 Apr  8 15:59 mounts
lrwxrwxrwx   1 root    root          0 Apr  8 15:59 root -> /
-r--r--r--   1 root    root          0 Apr  8 15:59 stat
-r--r--r--   1 root    root          0 Apr  8 15:59 statm
-r--r--r--   1 root    root          0 Apr  8 15:59 status
/proc/2461/fd:
total 0
dr-x-----   2 root    root          0 Apr  8 15:59 .
dr-xr-xr-x   3 root    root          0 Apr  8 15:59 ..
lrwx-----   1 root    root        64 Apr  8 15:59 0 -> /dev/pts/1
lrwx-----   1 root    root        64 Apr  8 15:59 1 -> /dev/pts/1
lrwx-----   1 root    root        64 Apr  8 15:59 10 -> socket:[75927]
lrwx-----   1 root    root        64 Apr  8 15:59 11 -> socket:[75930]
lrwx-----   1 root    root        64 Apr  8 15:59 12 -> socket:[75932]
lrwx-----   1 root    root        64 Apr  8 15:59 13 -> socket:[75935]
lrwx-----   1 root    root        64 Apr  8 15:59 14 -> socket:[75939]

```

#more `find /proc` > /mnt/floppy/moreproc.txt

Output (Truncated to /proc/2461/cmdline to save space):

/proc/2461/cmdline:

gedit

The command **lsdf** lists open files. The “d rtd” option tells it to list by file descriptor type “rtd.” This lists processes that are running “chroot’d” and what directory that are now rooted in.

lsdf -d rtd > /mnt/floppy/lsdf-rtd.txt

Output:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
init	1	root	rtd	DIR	3,2	4096	2	/
keventd	2	root	rtd	DIR	3,2	4096	2	/
kapmd	3	root	rtd	DIR	3,2	4096	2	/
ksoftirqd	4	root	rtd	DIR	3,2	4096	2	/
kswapd	5	root	rtd	DIR	3,2	4096	2	/
bdfldsh	6	root	rtd	DIR	3,2	4096	2	/
kupdated	7	root	rtd	DIR	3,2	4096	2	/
mdrecover	8	root	rtd	DIR	3,2	4096	2	/
kjournald	12	root	rtd	DIR	3,2	4096	2	/
khubb	68	root	rtd	DIR	3,2	4096	2	/
kjournald	161	root	rtd	DIR	3,2	4096	2	/
syslogd	407	root	rtd	DIR	3,2	4096	2	/
klogd	411	root	rtd	DIR	3,2	4096	2	/
portmap	428	rpc	rtd	DIR	3,2	4096	2	/
rpc.statd	447	rpcuser	rtd	DIR	3,2	4096	2	/
cardmgr	503	root	rtd	DIR	3,2	4096	2	/
apmd	541	root	rtd	DIR	3,2	4096	2	/
sshd	630	root	rtd	DIR	3,2	4096	2	/
xinetd	677	root	rtd	DIR	3,2	4096	2	/
lpd	691	lp	rtd	DIR	3,2	4096	2	/
sendmail	711	root	rtd	DIR	3,2	4096	2	/

sendmail	721	smmsp	rtd	DIR	3,2	4096	2	/
gpm	731	root	rtd	DIR	3,2	4096	2	/
crond	740	root	rtd	DIR	3,2	4096	2	/
xfs	771	xfs	rtd	DIR	3,2	4096	2	/
atd	789	daemon	rtd	DIR	3,2	4096	2	/
mingetty	798	root	rtd	DIR	3,2	4096	2	/
mingetty	799	root	rtd	DIR	3,2	4096	2	/
mingetty	800	root	rtd	DIR	3,2	4096	2	/
mingetty	801	root	rtd	DIR	3,2	4096	2	/
mingetty	802	root	rtd	DIR	3,2	4096	2	/
mingetty	803	root	rtd	DIR	3,2	4096	2	/
gdm-binar	804	root	rtd	DIR	3,2	4096	2	/
gdm-binar	849	root	rtd	DIR	3,2	4096	2	/
X	850	root	rtd	DIR	3,2	4096	2	/
gnome-ses	859	root	rtd	DIR	3,2	4096	2	/
ssh-agent	917	root	rtd	DIR	3,2	4096	2	/
gconfd-2	928	root	rtd	DIR	3,2	4096	2	/
esd	930	root	rtd	DIR	3,2	4096	2	/
bonobo-ac	939	root	rtd	DIR	3,2	4096	2	/
metacity	941	root	rtd	DIR	3,2	4096	2	/
gnome-set	943	root	rtd	DIR	3,2	4096	2	/
fam	947	root	rtd	DIR	3,2	4096	2	/
gnome-pan	954	root	rtd	DIR	3,2	4096	2	/
nautilus	956	root	rtd	DIR	3,2	4096	2	/
magicdev	958	root	rtd	DIR	3,2	4096	2	/
pam-panel	961	root	rtd	DIR	3,2	4096	2	/
rhn-apple	963	root	rtd	DIR	3,2	4096	2	/
pam_times	964	root	rtd	DIR	3,2	4096	2	/
nautilus	965	root	rtd	DIR	3,2	4096	2	/
nautilus	966	root	rtd	DIR	3,2	4096	2	/
nautilus	967	root	rtd	DIR	3,2	4096	2	/
nautilus	968	root	rtd	DIR	3,2	4096	2	/
gnome-ter	972	root	rtd	DIR	3,2	4096	2	/
bash	973	root	rtd	DIR	3,2	4096	2	/
bash	2397	root	rtd	DIR	3,2	4096	2	/
lsof	2444	root	rtd	DIR	3,2	4096	2	/
lsof	2445	root	rtd	DIR	3,2	4096	2	/
bash	26590	root	rtd	DIR	3,2	4096	2	/
man	27908	root	rtd	DIR	3,2	4096	2	/
sh	27909	root	rtd	DIR	3,2	4096	2	/
sh	27910	root	rtd	DIR	3,2	4096	2	/
less	27915	root	rtd	DIR	3,2	4096	2	/

The **top** command lists processes in order of CPU usage. The “b” option specifies batch mode so it does not accept command line input and the “n1” option specifies one iteration then an exit. We just take a snapshot of what processes are currently using the most CPU time. This may be helpful in determining what is happening to the system.

top -b -n1 > /mnt/floppy/top.txt

Output:

```
2:38pm up 2:18, 1 user, load average: 0.27, 0.21, 0.33
62 processes: 61 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 26.4% user, 5.3% system, 2.5% nice, 65.6% idle
Mem: 320080K av, 304700K used, 15380K free, 0K shrd, 9476K buff
Swap: 650152K av, 0K used, 650152K free 229932K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
850	root	5	-10	17708	9256	3792	S<	3.8	2.8	25:39	X
2446	root	15	0	1060	1060	828	R	2.8	0.3	0:00	top
958	root	15	0	5704	5700	4800	S	1.9	1.7	0:31	magicdev

1	root	15	0	480	480	428	S	0.0	0.1	0:04	init
2	root	15	0	0	0	0	SW	0.0	0.0	0:00	keventd
3	root	15	0	0	0	0	SW	0.0	0.0	0:00	kapmd
4	root	34	19	0	0	0	SWN	0.0	0.0	0:00	ksoftirqd_CPU0
5	root	15	0	0	0	0	SW	0.0	0.0	0:00	kswapd
6	root	25	0	0	0	0	SW	0.0	0.0	0:00	bdflood
7	root	15	0	0	0	0	SW	0.0	0.0	0:00	kupdated
8	root	24	0	0	0	0	SW	0.0	0.0	0:00	mdrecoveryd
12	root	15	0	0	0	0	SW	0.0	0.0	0:00	kjournald
68	root	16	0	0	0	0	SW	0.0	0.0	0:00	khubb
161	root	15	0	0	0	0	SW	0.0	0.0	0:00	kjournald
407	root	15	0	536	536	456	S	0.0	0.1	0:00	syslogd
411	root	15	0	428	428	376	S	0.0	0.1	0:00	klogd
428	rpc	15	0	532	532	460	S	0.0	0.1	0:00	portmap
447	rpcuser	16	0	728	728	640	S	0.0	0.2	0:00	rpc.statd
503	root	15	0	644	644	488	S	0.0	0.2	0:00	cardmgr
541	root	15	0	476	476	428	S	0.0	0.1	0:00	apmd
630	root	17	0	1468	1468	1224	S	0.0	0.4	0:00	sshd
677	root	15	0	904	904	772	S	0.0	0.2	0:00	xinetd
691	lp	15	0	1072	1072	924	S	0.0	0.3	0:00	lpd
711	root	15	0	2268	2268	1664	S	0.0	0.7	0:00	sendmail
721	smmsp	15	0	2052	2048	1560	S	0.0	0.6	0:00	sendmail
731	root	15	0	428	428	380	S	0.0	0.1	0:00	gpm
740	root	15	0	612	612	540	S	0.0	0.1	0:00	crond
771	xfs	15	0	3244	3244	884	S	0.0	1.0	0:00	xfs
789	daemon	15	0	520	520	464	S	0.0	0.1	0:00	atd
798	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
799	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
800	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
801	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
802	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
803	root	15	0	404	404	356	S	0.0	0.1	0:00	mingetty
804	root	15	0	2936	2936	2816	S	0.0	0.9	0:00	gdm-binary
859	root	15	0	8104	8100	6212	S	0.0	2.5	0:00	gnome-session
917	root	15	0	992	992	800	S	0.0	0.3	0:00	ssh-agent
928	root	15	0	4376	4376	1908	S	0.0	1.3	0:01	gconfd-2
930	root	15	0	1656	1656	368	S	0.0	0.5	0:00	esd
939	root	15	0	2200	2200	1804	S	0.0	0.6	0:00	bonobo-activati
941	root	15	0	6132	6132	5060	S	0.0	1.9	0:01	metacity
943	root	15	0	7224	7220	5732	S	0.0	2.2	0:01	gnome-settings-
947	root	15	0	1268	1268	1040	S	0.0	0.3	0:00	fam
954	root	15	0	10460	10M	8124	S	0.0	3.2	0:02	gnome-panel
956	root	15	0	14240	13M	9440	S	0.0	4.4	0:05	nautilus
961	root	15	0	3844	3844	3240	S	0.0	1.2	0:00	pam-panel-icon
963	root	15	0	12596	12M	8576	S	0.0	3.9	0:56	rhn-applet-gui
964	root	15	0	476	476	416	S	0.0	0.1	0:00	pam_timestamp_c
972	root	15	0	9616	9612	6764	S	0.0	3.0	6:01	gnome-terminal
973	root	15	0	1536	1536	1356	S	0.0	0.4	0:04	bash
26590	root	15	0	1512	1512	1152	S	0.0	0.4	0:00	bash
27908	root	16	0	736	736	448	S	0.0	0.2	0:00	man
27909	root	17	0	964	964	888	S	0.0	0.3	0:00	sh
27915	root	15	0	616	616	500	S	0.0	0.1	0:00	less

Having captured information about the currently running processes that will be lost with a reboot or power-down, we move on to the next volatile set of information; the network connections.

The **netstat** command lists network connections. The “a” option specifies both active connections and those ports that are listening or open for a connection. This can often be a superb clue as to whether or not back doors have been installed. The “p” option (not supported by Solaris or a few other Unix versions) will list which processes are holding the ports open. This can be the clue to identify which process is a backdoor. The “inet” option specifies raw, udp, and tcp socket information.

```
netstat -ap --inet > /mnt/floppy/netstat.txt
```

Output:

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
-------	--------	--------	---------------	-----------------	-------	------------------

tcp	0	0	*:32768	.*	LISTEN	447/rpc.statd
tcp	0	0	localhost.localdo:	32769 *.*	LISTEN	677/xinetd
tcp	0	0	*:printer	.*	LISTEN	691/lpd Waiting
tcp	0	0	*:sunrpc	.*	LISTEN	428/portmap
tcp	0	0	*:x11	.*	LISTEN	850/X
tcp	0	0	*:ssh	.*	LISTEN	630/ssh
tcp	0	0	localhost.localdom:	smtp *.*	LISTEN	711/sendmail: accep
udp	0	0	*:32768	.*		447/rpc.statd
udp	0	0	*:sunrpc	.*		428/portmap

This next command will also list which processes are holding open which network ports. This can be useful on systems that do not support the “p” option of **netstat**. The **lsof** command lists open files and the “i” option specifies the files that designate open network connections. The “+M” option will list any portmapping associated with the open ports. This can help explain what the ports are used for and whether or not they should be of any concern.

```
lsof +M -i > /mnt/floppy/lsof-Mi.txt
```

Output:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
portmap	428	rpc	3u	IPv4	828	UDP	*:sunrpc[portmapper]	
portmap	428	rpc	4u	IPv4	829	TCP	*:sunrpc[portmapper]	(LISTEN)
rpc.statd	447	rpcuser	4u	IPv4	896	UDP	*:32768[status]	
rpc.statd	447	rpcuser	6u	IPv4	899	TCP	*:32768[status]	(LISTEN)
sshd	630	root	3u	IPv4	1358	TCP	*:ssh	(LISTEN)
xinetd	677	root	5u	IPv4	1420	TCP	localhost.localdomain:32769[sgi_fam]	(LISTEN)
lpd	691	lp	6u	IPv4	1455	TCP	*:printer	(LISTEN)
sendmail	711	root	4u	IPv4	1517	TCP	localhost.localdomain:smtp	(LISTEN)
X	850	root	1u	IPv4	1706	TCP	*:x11	(LISTEN)
fam	947	root	0u	IPv4	1420	TCP	localhost.localdomain:32769[sgi_fam]	(LISTEN)
fam	947	root	1u	IPv4	1420	TCP	localhost.localdomain:32769[sgi_fam]	(LISTEN)
fam	947	root	2u	IPv4	1420	TCP	localhost.localdomain:32769[sgi_fam]	(LISTEN)

The **ifconfig** command will list the configuration of all network interfaces. This could be useful in determining if additional virtual interfaces have been set up by any hackers or are listening on unexpected IP addresses. Additionally, the **ifconfig** command will show the “PROMISC” flag. This would indicate that the interface is in promiscuous mode, which would be used to sniff network traffic (i.e. for passwords).

```
ifconfig > /mnt/floppy/ifconfig.txt
```

Output:

```
eth0  Link encap:Ethernet  HWaddr 00:00:86:3D:18:87
      inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTI CAST  MTU:1500  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:7 errors:0 dropped:0 overrun s:0 carrier:4
      collisions:0 txqueuelen:100
      RX bytes:0 (0.0 b)  TX bytes:294 (294.0 b)
      Interrupt:3 Base address:0x300

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:139 errors:0 dropped:0 overruns:0 frame:0
      TX packets:139 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:9448 (9.2 Kb)  TX bytes:9448 (9.2 Kb)
```

Now that we have captured some information about the network connections and setup that will be lost after reboot, let us take a look at who might be on the system. The information about who is currently using the system will definitely be lost if we reboot.

The **who** command lists the people currently logged on to the system. The "H" option specifies to include the column titles in the output to make it easier to read and the "i" option specifies to include the idle time for the users. This will help us decide if the user listed is just associated with a crashed program or is actually active on the system.

```
who -Hi > /mnt/floppy/who.txt
Output:
root      :0               Apr  8 12:22
root
# users=1
```

The **finger** command also displays information about the system's users but it is not limited to those who are currently logged in. It will list home directories and other information for the user as well. While this may not give us a lot of information about what may be currently happening on the system, it will help us identify any new or bogus user accounts that may have been created.

```
finger -ls > /mnt/floppy/finger.txt
Output:
Login: root                                     Name: root
Directory: /root                               Shell: /bin/bash
On since Tue Apr  8 12:22 (PDT) on :0 (messages off)
Mail last read Tue Apr  8 13:53 2003 (PDT)
No Plan.
```

The **last** command will give us some information on who has recently been logged on to the system. From the man page **last** (1), "**Last** searches back through the file /var/log/wtmp ... and displays a list of all users logged in (and out) since that file was created." The "a" option specifies displaying the hostname of the user in case it was a remote login, the "i" option specifies displaying the IP address of any remote hosts users logged in from, the "d" option specifies translating the IP address back into a hostname, and the "x" option will display system shutdowns and run level changes. The **lastb** command is the same but looks in /var/log/btmp to display information about failed login attempts. Note that the administrator would have needed to create /var/log/btmp to log this info. While this information will be useful, it must be taken with a grain of salt. Since the two commands read files that reside on the system, the intruder may have modified those files. However, if the files have not been modified, which would take a fairly sophisticated root kit or hacker, then this information can be invaluable.

```
last -aix > /mnt/floppy/last.txt
Output:
wtmp begins Tue Apr  8 13:28:40 2003
lastb -aix > /mnt/floppy/lastb.txt
```

Output:
No such file or directory.

The script has collected information about the processes that are running, the network connections, and the system's users. The last evidence collecting command in the script is the **lsof** command. This command lists open files on the system. The first command with the "+L1" option tells **lsof** to list open files that have a link count less than one. In Unix-type operating systems, you can have multiple links to the same file. Each link will increase the link count. When you delete all links to the file, the link count becomes zero (less than one). This search is really only useful for finding one thing. A hacker might have a program logging information (keystrokes for example). After the program opens the file for logging, the hacker removes the last link to the file (e.g. **rm logfile**). Since a program has the file open for writing, it still exists on the system with a link count of zero. Its data is still accessible but the file will not be shown in any listing of files. The "+L1" option for **lsof** will find it. I would examine closely any results that this command gives.

lsof +L1 > /mnt/floppy/lsofL1.txt

Output:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NLINK	NODE	NAME
cardmgr	503	root	1u	CHR	254,0		0	376309	/var/lib/pcmcia/cm-503-0
(deleted)									
cardmgr	503	root	2u	CHR	254,1		0	376310	/var/lib/pcmcia/cm-503-1
(deleted)									
X	850	root	mem	DEL	0,4		0	131072	/SYSV00000000
X	850	root	mem	DEL	0,4		0	163841	/SYSV00000000
X	850	root	mem	DEL	0,4		0	196610	/SYSV00000000
X	850	root	mem	DEL	0,4		0	229379	/SYSV00000000
X	850	root	mem	DEL	0,4		0	262148	/SYSV00000000
X	850	root	mem	DEL	0,4		0	294917	/SYSV00000000
X	850	root	mem	DEL	0,4		0	327686	/SYSV00000000
X	850	root	mem	DEL	0,4		0	360455	/SYSV00000000
X	850	root	mem	DEL	0,4		0	393224	/SYSV00000000
X	850	root	mem	DEL	0,4		0	458761	/SYSV00000000
gnome-ses	859	root	mem	DEL	0,4		0	131072	/SYSV00000000
gnome-ses	859	root	mem	DEL	0,4		0	163841	/SYSV00000000
metacity	941	root	mem	DEL	0,4		0	229379	/SYSV00000000
gnome-set	943	root	mem	DEL	0,4		0	196610	/SYSV00000000
gnome-pan	954	root	mem	DEL	0,4		0	327686	/SYSV00000000
nautilus	956	root	mem	DEL	0,4		0	262148	/SYSV00000000
nautilus	956	root	mem	DEL	0,4		0	294917	/SYSV00000000
rhnp-apple	963	root	mem	DEL	0,4		0	360455	/SYSV00000000
nautilus	965	root	mem	DEL	0,4		0	262148	/SYSV00000000
nautilus	965	root	mem	DEL	0,4		0	294917	/SYSV00000000
nautilus	966	root	mem	DEL	0,4		0	262148	/SYSV00000000
nautilus	966	root	mem	DEL	0,4		0	294917	/SYSV00000000
nautilus	967	root	mem	DEL	0,4		0	262148	/SYSV00000000
nautilus	967	root	mem	DEL	0,4		0	294917	/SYSV00000000
nautilus	968	root	mem	DEL	0,4		0	262148	/SYSV00000000
nautilus	968	root	mem	DEL	0,4		0	294917	/SYSV00000000
gnome-ter	972	root	mem	DEL	0,4		0	393224	/SYSV00000000
gedit	2461	root	mem	DEL	0,4		0	458761	/SYSV00000000

As you can see, there are a few listed on the machine. Most appear to be temp files in use by programs like gedit and nautilus. If there were anything suspicious, there are other tools available that may help you identify what is stored at those inodes (see TCT at the end of the paper).

The second **lsof** command simply lists all open files opened by all active processes. This information will be useful to identify not only what is running on the system but also what those processes might be doing. It

would help you identify log files and other files that might give you information about what the intruders might be doing.

```
lsof > /mnt/floppy/lsof.txt
```

Output (note, the actual output was 83 pages long, this is a sample):

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
init	1	root	cwd	DIR	3,2	4096	2	/
init	1	root	rtd	DIR	3,2	4096	2	/
init	1	root	txt	REG	3,2	33960	146801	/sbin/init
init	1	root	mem	REG	3,2	87341	162606	/lib/ld-2.2.93.so
init	1	root	mem	REG	3,2	1395734	634002	/lib/i686/libc-2.2.93.so
init	1	root	10u	FIFO	3,2		65055	/dev/initctl
keventd	2	root	cwd	DIR	3,2	4096	2	/
keventd	2	root	rtd	DIR	3,2	4096	2	/
keventd	2	root	10u	FIFO	3,2		65055	/dev/initctl
kapmd	3	root	cwd	DIR	3,2	4096	2	/
kapmd	3	root	rtd	DIR	3,2	4096	2	/
kapmd	3	root	10u	FIFO	3,2		65055	/dev/initctl
ksoftirqd	4	root	cwd	DIR	3,2	4096	2	/
ksoftirqd	4	root	rtd	DIR	3,2	4096	2	/
ksoftirqd	4	root	10u	FIFO	3,2		65055	/dev/initctl
kswapd	5	root	cwd	DIR	3,2	4096	2	/
kswapd	5	root	rtd	DIR	3,2	4096	2	/
kswapd	5	root	10u	FIFO	3,2		65055	/dev/initctl
bdflush	6	root	cwd	DIR	3,2	4096	2	/
bdflush	6	root	rtd	DIR	3,2	4096	2	/
bdflush	6	root	10u	FIFO	3,2		65055	/dev/initctl
kupdated	7	root	cwd	DIR	3,2	4096	2	/
kupdated	7	root	rtd	DIR	3,2	4096	2	/
kupdated	7	root	10u	FIFO	3,2		65055	/dev/initctl

Now that the script is done collecting evidence, it cleans up a bit. It records the date and time at which it stopped collecting evidence. Then the **more** command appends all of the files containing the results of the commands into one big file called results.txt. This will make it easy to keep all the evidence together. Next the script generates a md5 checksum of the results.txt file so we can ensure and prove later that the evidence has not been modified since it was gathered.

```
date > /mnt/floppy/stopdate.txt
```

Output:

```
Tue Apr 8 14:38:59 PDT 2003
```

```
more *.txt > /mnt/floppy/results.txt
```

Output:

This file merely contains all of the above listed output in one easy to use file. It is not repeated here.

```
md5sum results.txt > /mnt/floppy/resultsmd5.txt
```

```
e9b9845d95ab4f69becb3d8ddcdb3bd3 results.txt
```

Script Usage

Due to the preparation work, use of the script is fairly easy. The first thing needed is to mount the response CDROM and the evidence floppies. This can be done with the following commands:

```
mount /dev/cdrom /mnt/cdrom
```

```
mount /dev/fd0 /mnt/floppy
```

Next you run the trusted shell on the CDROM:

/mnt/cdrom/bash

Lastly, you run the script itself:

/mnt/cdrom/script

That is all that is needed. The script does the rest. Once the script has completed, you handle the evidence floppies in a forensically controlled manner and proceed with the investigation. The great advantage you have now, is that having collected evidence about what processes are running, who was logged on and connected to the system, and what files were open, you stand to lose much less if you disconnect or power down the system. You are in a much better position to respond to management or operational demands. You have not limited your future actions.

The original intent was to write a script that could be used across Unix operating systems and platforms (Solaris, BSD, Red Hat...). However, there are some difficulties in that goal. The first is that the **lsbf** command (used three times in the script) uses very basic system calls. Therefore, **lsbf** would likely need to be compiled for every operating system and platform that it needs to be used on. This alone would create a need for a separate trusted CDROM for each system. Another difference between operating systems is the `LD_LIBRARY_PATH` variable. The name of the appropriate variable changes from system type to system type.

One way to approach this would be to have the script use the results of a **uname** command (be sure to include a trusted **uname** and any required libraries on your CDROM) to determine what system it's running on. Then the script could access the appropriate bin and lib directories on your CDROM where you had the appropriate tools for each system. As written, the script is very close to being portable. Certainly the steps and philosophy would apply to any system.

Back to the Real World Example

My boss asked how long it would take to do an investigation and how long it would take to wipe the disk clean and install a secure operating system. I told him it would take several hours to begin to investigate what was happening. We would need to do some research, gather the tools together, and likely fly an IT department person to the branch office to do the investigation. I then told him I could get the disk cleaned, a new operating system installed, secured, and back in production in 4 hours or so. We discussed how critical the data on the FTP server was itself. My boss's exact words were "I don't want to be a cop here..." I had the user shutdown the machine. I then talked him through wiping the disk clean, and installing the latest version of Red Hat Linux on a different IP address. I then connected remotely with SSH and further secured the machine. It was added to the servers that the IT department is responsible for maintaining.

If I had to guess, I would say it was probably being used to share files between the hacker and his friends that had compromised the machine. However, we will never know. It sure would be good to know if the attacker was local to our corporation or indeed a stranger on the Internet. It would be nice to know if it was being used in attacks on other machines. If I had the initial response CDROM ready, I could have had better answers for my boss in a few minutes about what was happening and whether a further investigation was needed.

The script in its entirety

```
#Set environment variables and establish trusted shell
PATH="/mnt/cdrom/bin"
LD_LIBRARY_PATH="/mnt/cdrom/lib"
export PATH
export LD_LIBRARY_PATH

date > startdate.txt

ls -al /mnt/floppy > /mnt/floppy/startdirempty.txt
echo $PATH > /mnt/floppy/path.txt
echo $LD_LIBRARY_PATH > /mnt/floppy/libpath.txt
ldd /mnt/cdrom/bin/* > /mnt/floppy/ldd.txt
md5sum /mnt/cdrom/script > /mnt/floppy/scriptmd5.txt

ps -auxww > /mnt/floppy/ps.txt
#ls -alR /proc > /mnt/floppy/ls-proc.txt
#more `find /proc` > /mnt/floppy/moreproc.txt
lsof -d rtd > /mnt/floppy/lsof-rtd.txt
top -b -n1 > /mnt/floppy/top.txt

netstat -ap --inet > /mnt/floppy/netstat.txt
lsof +M -i > /mnt/floppy/lsof-Mi.txt
ifconfig > /mnt/floppy/ifconfig.txt
who -Hi > /mnt/floppy/who.txt
finger -ls > /mnt/floppy/finger.txt
last -aix > /mnt/floppy/last.txt
lastb -aix > /mnt/floppy/lastb.txt

lsof > /mnt/floppy/lsof.txt

date > /mnt/floppy/stopdate.txt

more *.txt > /mnt/floppy/results.txt
md5sum results.txt > /mnt/floppy/resultsmd5.txt
```

Structure of CDROM

Note: "Trusted" versions ideally come from a freshly installed system or from the source media. There is some latitude, however. You may decide that a tool compiled from the source can be trusted. Or you could designate tools obtained directly from software vendors as trusted. However, since there have been many cases of web sites distributing trojanized software (due to the site being hacked), you have to be careful with blindly trusting a site. Many sites will include an MD5 checksum provided by the developer to prove that it is a trusted version. The end result is that you must be able to prove (in

court perhaps) that the versions on your CDROM can be trusted over those on the compromised system. Perhaps the easiest way to create this CDROM, would be to follow the following steps:

1. Install Red Hat 8.0 on an absolutely clean system. The steps involved in installing Red Hat is beyond the scope of this paper. Red Hat's web site has detailed instructions. Use either a packaged version of Red Hat purchased in a store or a downloaded version that was verified using the MD5 checksum provided by Red Hat. This will ensure that the system is clean and has nothing but trusted code.
2. Create a directory on the system called /CDTOBURN.
3. Create the directory structure listed below and copy the listed files to them.
4. If the Red Hat system does not have a CDRW, just copy the CDTOBURN directory to a floppy or over the network to a machine that does and burn it. Then the CD should be ready to go!

The root directory of the CDROM contains:

- bash (a trusted bash shell executable file)
- script (a file containing the script listed above)
- bin (a directory containing the trusted versions of our tools' binaries)
- lib (a directory containing the trusted versions of the shared libraries required by our tools)

The bin directory contains the following trusted versions of the tools:

- date
- echo
- finger
- ifconfig
- last
- ldd
- lddlibc4
- ls
- lsof
- md5sum
- more
- netstat
- ps
- rm
- top
- who

The lib directory contains the following trusted versions of shared libraries:

- libacl.so.1
- libattr.so.1
- libc.so.6
- libncurses.so.5

- libproc.so.2.0.7
- libpthread.so.0
- librt.so.1
- libtermcap.so.2

The Coroner's Toolkit (TCT)

The TCT can be found on the web at the URL listed with the references. On that page it describes it as “a collection of programs by Dan Farmer and Wietse Venema for a post-mortem analysis of a UNIX system after break-in.” Like this script, it attempts to gather the evidence “in order of volatility” as Farmer and Venema say. As their toolset has some of the same goals as the script presented in this paper, a comparison is in order. Here are a few differences between the TCT and this script.

TCT has a much broader scope than this script. While this script is focusing on collecting only that data that will be lost if you power down or disconnect the system, TCT collects much more information. TCT collects information about the file systems and how much disk space is free, information about free inodes, the MD5 checksum of all files, the bash history of users, and much more. Some of this information would be retrievable after you shutdown or disconnect the system. This paper only tries to collect evidence that would be lost.

By its own admission TCT is a complex set of tools. This paper was designed to be relatively straight forward and easy to use and follow. Also, the vast majority of the evidence collection in TCT is done by a program called “grave-robber.” It is not easy to customize which commands run in what order and with what options, etc... The script in this paper is much easier to modify as all the commands are out in the open for everyone to see. This may make modification to a specific environment easier.

TCT is very much like the script in this paper in that it takes into account the volatility of the data it is collecting. It also uses MD5 to generate checksums to ensure that the evidence has not been tampered with since collection. It records the timestamp information as it acts.

While there is definitely some overlap, the focus of the two tools is different. This paper only focuses on trying to collect the data that will be lost if management orders a shutdown/disconnection of the system. A more detailed analysis takes more time and more tools and can be done later. TCT is a great toolset and deserves a look.

List of References

- Acheson, Steve. Green, John. Pomeranz, Hal. Track 6 - UNIX Security Tools. 6.3 Topics in UNIX Security. Bethesda: SANS Institute, 2002. Course Materials.
- Pomeranz, Hal. Track 6 - UNIX Security Tools. 6.2 UNIX Security Tools. Bethesda: SANS Institute, 2002. Course Materials.

- Mandia, Kevin. Prosise, Chris. Investigating Computer Security. Las Vegas: Seminar at Interop2002, May 2002. Seminar Materials.
- "Creating a Computer Security Incident Response Team: A Process for Getting Started." Carnegie Mellon CERT Coordination Center. URL: <http://www.cert.org/csirts/Creating-A-CSIRT.html> (8 Apr. 2003)
- "The MD5 Message-Digest Algorithm." Internet Engineering Task Force Request for Comments 1321. April 1992. URL: <http://www.ietf.org/rfc/rfc1321.txt?number=1321> (4 Jan. 2003)
- Welsh, Matt. Dalheimer, Matthias. Kaufman, Lar. Running Linux – Third Edition. O'Reilly, 1999.
- Pomeranz, Hal. "Static Linking Under Solaris." 2001. URL: <http://www.deer-run.com/~hal/sol-static.txt> (4 Jan. 2003)
- "Red Hat Linux 8.0 Manual Pages." Red Hat, Inc. 2002
- Dan Farmer and Wietse Venema. "The Coroner's Toolkit (TCT)." URL: <http://www.porcupine.org/forensics/tct.html> (8 Apr. 2003).

© SANS Institute 2003, Author retains full rights.