;login:

THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus Issue: Security

Guest Editor: Rik Farrow

inside:

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

forensics lite

by Brad Powell

Brad Powell works as a senior network security consultant for Sun Microsystems Professional Services, where he does security research and writes security tools.

brad.powell@sun.com

As the old Boy Scout motto goes, "Be Prepared." Preparation is the key to successfully surviving a security incident. If your organization hasn't thought about how you are going to handle that future security incident, then NOW is the time to start thinking about it.

Let's discuss some reasons why. When I was working internal security for Sun, we figured out four or five likely security-incident scenarios and laid plans on how they would be handled and coordinated. The plans didn't all survive the real-life incidents, and some ended up being changed on the fly to deal with things we didn't expect, but all in all, they allowed us to bring the right resources to the problem. Having a plan gave us the ability to work through all the surprises while still keeping our sanity – a good reason to have one in this day and age.

Hoever, in my experience, most sites are not prepared and don't have a plan in place, so we will address that scenario in detail.

How to Respond to an Attack

So you think you have been hacked and don't know how to respond. First off, don't PANIC. I've handled plenty of intrusions and a large percentage of them are false alarms. Secondly, by the time you have figured out that you truly have been hacked, it's probably too late to panic, so I'd advise skipping the panic step altogether.

So what do you do? Well, the course is often dictated to you by management. Your chances of catching the intruder are pretty small, so you might ask yourself "Why bother?" Management, who may be on your case to get the resource back up and running in a hurry, doesn't want to spend the time going through a full forensic analysis; all they want is that Web server "Back on the Net so we can conduct business." If this is an all too familiar scenario, I beg you, when you get done reading this article, take it to your management and have them read it too. Things will never get better until your organization starts prosecuting and handling incidents.

We seem to have two opposing problems. Management wants and needs to get the system back up and running quickly, but doing forensic analysis takes a lot of time. I submit to you that just getting the system back up on the Net begs the attacker to hit you again and again. Until you fix the problems, you're living on borrowed time.

Here is an example. When working internal security for Sun Microsystems, we had an incident involving Kevin Mitnick. It was only because of a Herculean effort by Tsutomu Shimomura that Mitnick was captured, but it was because Sun (and others) collected so much evidence that Mitnick pleaded guilty to the charges. The point is that the evidence you collect may not be readily used by your site, but it still has value. My copy of the evidence as well as the original disk drives sat in a safe gathering dust for over three years before the evidence was ever seen in court.

All well and good, but what if you don't know if the system has been compromised? Well your Network IDS and local Tripwire data will tell you what has occurred and what has changed. What? You don't have either of those? I'm not surprised. Now you have a decision to make. Again, it's probably based on resources. At this point you can:

- Call in a professional to examine the tampered disk and give you an analysis of what went wrong.
- Turn over the evidence to law enforcement.
- Look at the disk yourself and see what you can learn from it.

- Forget the whole thing.

Sadly, the last option is taken all too often. I beg you to reconsider before taking this option.

Since we are discussing "forensics lite," I'll leave the first two options for another article, or for others to cover, and will focus on what you can learn from the information to ensure that you really have plugged the hole, so you don't get any repeat incidents, and to ensure you have cleaned up the mess.

Our first step is to make sure we really have been hacked. This can be trickier than you think. Why? Well, to do it right, to avoid tampering with the evidence, you have to follow all the procedures as if you are sure you have been hacked. So we assume we have been hacked even though we are still unsure. First, take the system offline. Then, to preserve the memory and running process tables, we need to suspend the system without doing a graceful shutdown using `shutdown(1)` or `init 0`. With a Solaris system, we use the Stop-A or L1-A sequence. Check with your operating system vendor for equivalent functionality. Then we mount the file system onto another system, make a full backup, preserve the original disk if at all possible, and build a new system disk that is hardened against attack. Do NOT reuse the system backup you just made to restore the system, since that is evidence, and most likely will only reintroduce the old bug back into the system if used. That system cannot be trusted.

Our first step is to make sure we really have been hacked. This can be trickier than you think.

This is the safest way, and even if you find out later that you have not been hacked and discover this was a false alarm, you have still improved your system security. Consider the fresh install an added bonus and a worthy exercise.

You may later have to reload portions of the suspect system backup, but what gets reloaded should not be the operating system or any other component that you can get from distribution media. Make sure to apply all the latest patches after rebuilding from distribution media.

The backup and fresh install gets the organization back up on the network and conducting business and gives you the opportunity to figure out what went wrong and how you can prevent it in the future. It also gives you, if you do discover evidence that may lead to prosecution, a clean copy of all evidence on the original system disk and on a backup tape.

From here on, I'm assuming we are working with a copy of the data and not with the original. For full details on doing full forensics and how to make a copy of the disk using `dd(1)` for your forensic lite, refer to TCT, The Coroner's Toolkit (*http://www.fish.com/tct/*), or "DD and Computer Forensics," a simple guide to using the `dd(1)` command, by Thomas Rude (*http://www.crazytrain.com/dd.html*). We use the UNIX `dd(1M)` command instead of using `tar(1)` or `ufsdump(1M)` so that we preserve the byte-by-byte layout of the system disk and the swap area of the disk. This is important if we decide later that we need to do more in-depth forensics using TCT or another forensic tool.

Now that we have a working copy and have preserved the original, there are some simple things we can do to poke at the disk to see if the operating system has been modified. The most often seen problem is a rootkit. A rootkit is a replacement set of Trojan binaries that is used by intruders to hide their use of the system and used to install back-door access to your system for future use.

Since we cannot assume the disk we are looking at hasn't been trojaned or rootkitted, we don't boot it

Again, since we are talking about a lite version of forensics, I'm going to stick with basic and simple tools. Let's start with Sidekick. Sidekick.sh is a simple Bourne shell script that works with MD5 to check the signatures of files. Sidekick is available from *http://www.sun.com/security/* or from *http://www.sun.com/blueprints/tools/fingerprint_license.html*. Although written for use with the Solaris Operating System, sidekick.sh is portable enough to use on most any UNIX-like operating system. We use Sidekick in this example instead of going to an industry standard like Tripwire because Tripwire can only report modifications since the last time you ran it. If you had run Tripwire before all this occurred, then you would already know what has changed. You might want to think about using Tripwire in the future (it is a really good tool/product), but without the previous baseline to compare against, you can't figure out what has changed. So we use Sidekick to try and create this baseline. As a side note, Sidekick can be used to look at legacy systems that you can't take offline to verify that the system hasn't been trojaned in the past; and from that point on, Tripwire ran daily will give you a reasonable start at system-level intrusion detection. This isn't a full solution, just a start.

Sidekick.sh, as the name implies, is a simple companion tool that just aids you in the collection of MD5 signatures. When used on a file, MD5 produces a cryptographic checksum of the file. It is nearly impossible for two binary files (assuming their length is not 0) to produce the same MD5 checksum unless the content of the binaries match. Thus, if we have two files with identical names but different contents, the MD5 checksums will not match. Please read the Sidekick manual page that accompanies distribution of sidekick.sh for details. I worked hard to make sure the man page had useful information and limitations on its effectiveness.

Since we cannot assume the disk we are looking at hasn't been trojaned or rootkitted we don't boot it; instead, we mount the partitions and gather the MD5 checksums from there, or in the case of a system we can't take offline, we run sidekick.sh locally on the system using a statically linked MD5 program and take the resulting MD5 checksum output to another non-suspect system and check the MD5 checksums from there.

Some of the useful options for sidekick.sh include:

-R Specify an alternate root directory. This option is useful for chroot areas such as on DMZ Web servers, and can be used when you are able to mount the various partitions of the suspect disk onto another system.

-r Check all the files that are commonly rootkitted, comparing them against what Sun ships. This is the basic common Trojan check and should be performed before any other Sidekick option to verify that the find(1) command, at a minimum, hasn't been trojaned.

-S This option is used with any of the other options to run Sidekick standalone, that is, without invoking the Perl script that compares checksums (explained below).

-a This option checks all files and creates the MD5 checksum. This option is a poor man's version of L5 or Tripwire and can be used to check a legacy system and create a reasonable baseline to ensure the binaries that are currently on the system are the ones shipped by Sun for the entire system. Once a baseline is determined, the user can then run L5 or Tripwire on a regular basis looking for changes. See the warning above.

Caution should be taken when using this option, as there will be many false positives reported for any locally modified configuration files or binaries.

So a typical check might include running sidekick.sh -R /mnt/suspectdisk -a -S which collects an MD5 checksum of all the system files started at the mount point. sidekick.sh -r just collects the MD5 checksums of a subset of binaries that are commonly found in a rootkit.

Now that we have MD5 checksums of all the files, we can compare this list to a list of MD5 checksums from a known uncompromised system. Whatever doesn't match is either a patched binary that didn't come with the original vendor release or a Trojan.

In the case of Sun Microsystems and Solaris, the task of finding a listing of known good MD5 checksums has been made easier for you. Sun provides an MD5 fingerprint database of every binary that it has ever shipped. I strongly encourage you to get with your other vendors (Linux people, are you listening?) to have them supply MD5 checksums of every binary they ship. An additional tool called sfpC.pl is a Perl script that can be used with sidekick.sh to query Sun's Fingerprint database online. More information about the Sun Fingerprint database can be found by searching on the keyword "fingerprint database" at *http://www.sun.com/blueprints/* or by going directly to the Fingerprint database link at *http://sunsolve.Sun.COM/pub-cgi/fileFingerprints.pl*.

Sun provides an MD5 fingerprint database of every binary that it has ever shipped

An example use of sidekick.sh:

```
root# sidekick.sh -S -r
Operating in standalone mode, sfpC will not be run.
Searching for files commonly found in rootkits.
The output has been saved to rootkitfiles-md5.20010820130858
[Note: the example has been edited to 10 entries to conserve space]
root# cat rootkitfiles-md5.20010820130858
MD5 (/usr/bin/du) = 9b9d5b91bb697c0b5e8acdd7e8286b79
MD5 (/usr/bin/ls) = 351f5eab0baa6eddae391f84d0a6c192
MD5 (/usr/sbin/in.telnetd) = dae9a44a49faeaf54ddcb30578663b39
MD5 (/usr/bin/login) = b6915118b96ed4132f45db7332dcc293
MD5 (/usr/sbin/route) = a4fea6e7e07e377812773c8e71cc5f05
MD5 (/usr/sbin/inetd) = 90907143eb6a4909aee4a7297b5d12a7
MD5 (/usr/bin/passwd) = 39d9e82ed48669d6396fed0fb9c0f901
MD5 (/usr/sbin/in.rshd) = 9656d16a4550c925d00e78d90cb775c9
MD5 (/usr/sbin/in.rlogind) = 46c1c2ba01e36c8264a3d25c4097bc98
MD5 (/usr/sbin/syslogd) = 93e97f044f18c85bfe3a12fb77f1198e
```

We take this output file created by sidekick.sh called rootkitfiles-md5.20010820130858 to another system which has Internet connectivity, and feed it into sfpC.pl, the utility that queries Sun's Fingerprint database:

```
mysystem-> sfpC.pl rootkitfiles-md5.20010820130858

    9b9d5b91bb697c0b5e8acdd7e8286b79 - (/usr/bin/du) - 1 match(es)

canonical-path: /usr/bin/du
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
patch: 109803-01

351f5eab0baa6eddae391f84d0a6c192 - (/usr/bin/ls) - 1 match(es)
```

```
canonical-path: /usr/bin/ls
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

b6915118b96ed4132f45db7332dcc293 - (/usr/bin/login) - 1 match(es)

canonical-path: /usr/bin/login
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

a4fea6e7e07e377812773c8e71cc5f05 - (/usr/sbin/route) - 1 match(es)

canonical-path: /usr/sbin/route
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

90907143eb6a4909aee4a7297b5d12a7 - (/usr/sbin/inetd) - 1 match(es)

canonical-path: /usr/sbin/inetd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

39d9e82ed48669d6396fed0fb9c0f901 - (/usr/bin/passwd) - 1 match(es)

canonical-path: /usr/bin/passwd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

9656d16a4550c925d00e78d90cb775c9 - (/usr/sbin/in.rshd) - 1 match(es)

canonical-path: /usr/sbin/in.rshd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
patch: 108985-02

46c1c2ba01e36c8264a3d25c4097bc98 - (/usr/sbin/in.rlogind) - 1 match(es)

canonical-path: /usr/sbin/in.rlogind
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC

93e97f044f18c85bfe3a12fb77f1198e - (/usr/sbin/syslogd) - 1 match(es)

canonical-path: /usr/sbin/syslogd
package: SUNWcsu
version: 11.8.0,REV=2000.01.08.18.12
architecture: sparc
source: Solaris 8/SPARC
```

Now we need to examine the output from the Fingerprint database query. We notice that we received nine responses from the database; each response showed exactly one match, a reference of which file matched the files in the database, the OS release that match came from, and in some cases, the fact that some of the binaries were Sun patches and not directly tied to a distribution.

But wait! We sent the query 10 MD5 checksums. ONE didn't receive a return; in.telnetd was not found in Sun's Fingerprint database. What does this mean? This means that the in.telnetd on the system was never shipped by Sun. Is this a Trojan? Possibly. It could also mean that the user replaced in.telnetd with another version, possibly to add a feature that Sun doesn't provide, but either way in.telnetd needs to be looked at very closely to decide where it came from and if it is legitimate.

We have now looked at all the shipped binaries for a Solaris OS, but what if I am using Linux, or FreeBSD? Well some of the same tactics apply; it is just going to take additional steps. There are a few options. First, if we have another system which was installed using the same installation media and hasn't been modified too greatly by the user (not likely), we could build up our own MD5 database. A better option would be to install a fresh system using the installation media, install any updated RPMs (patches) that were previously applied (you do keep a log of all patches you have applied to each system, don't you?) then use it as root so we get every file:

```
root-on-clean-system# find / -type f -exec md5 {} \; >clean-md5db-ostype-date
```

Then you sort and unique the files to give an alphabetic listing of all files:

```
root-on-clean-system# sort -u clean-md5db-ostype-date >
                clean-md5db-ostype-date-sorted
```

Then we can do a simple compare of the two files, the MD5s collected from our suspect system with sidekick.sh -a and the one from a clean system.

```
root-on-suspect-system# sidekick.sh -S -a
Operating in standalone mode, sfpC will not be run.
Searching for all files commonly this option is used in conjunction with '-S'
The output has been saved to allfiles-md5
```

We then move or copy the allfiles-md5 to the clean system; run sort and unique there, and are ready to compare:

```
root-on-clean-system# sort -u allfiles-md5 >allfiles-md5-suspect
root-on-clean-system# diff allfiles-md5-suspect clean-md5db-ostype-
            date-sorted
```

This will give us a reference pointer and a starting point. This method will result in a lot of false negatives since system files such as the password file as well as any system files that have been modified will not match the MD5, but it will at least show us something meaningful.

Even though we are only doing the lite version of forensics, this is all getting complicated, isn't it? We could have saved ourselves a lot of grief up to this point if we had created an MD5 database as soon as we installed the system, and/or run something like Tripwire. An ounce of prevention is worth a pound of cure; or in this case, 20 minutes of MD5 up-front is worth 20 hours of MD5 later.

Have I scared you enough to get you to go run MD5 now before an intrusion? Good!

Even though we are only doing the lite version of forensics, this is all getting complicated, isn't it?

The first thing any intruder does is clean out their entries from the log files to mask or cloak their presence

Log Files

What about log files? Well they are worth reviewing, and if you have one local copy and one remote copy on your syslog server, then it should be a simple matter of comparing the log sizes to determine if your log files have shrunk in size. The first thing any intruder does is clean out their entries from the log files to mask or cloak their presence. Most rootkits have a built-in utility to do this. What? You only keep the log files locally and don't have a syslog server?

You are screwed. But, look at the log files anyway. Maybe it's a script kiddie, and their script failed to work properly due to a bad path to a utility or something, but don't bet on it. If you do have a syslog server set up, you are looking for anything in the remote file that doesn't match the local copy, and from there looking both before and after that entry for signs of the first penetration. Most intruders will have set up a back door that doesn't log their connections, so you need to see if there was any previous system scan or failed attempt before they found that bug in your application and introduced the rootkit into your system. This may or may not be worth the trouble. Lesson learned.

From here we need to look for anything which might have been loaded into the kernel as a module. The process table and /proc file system if your system has one (most modern UNIX systems do) are worth examining. Why? Well, after breaking into a system a clever intruder will install his or her back door and then delete their toolkit to keep it from being captured or examined. If you're lucky and have the time to do full forensics, it is quite possible to recover these deleted files. Take a look at the examples (shameless plug alert) from the Honeynet project (*http://project.honeynet.org*) or The Coroner's Toolkit. Recovery of files takes a long time and requires plenty of disk space to attempt, with no guarantee of results, but it is amazing just how pervasive data is and how hard it is to truly delete.

This may be too extensive for forensics lite, so we will try looking into the /proc file system first before attempting this.

From the man pages on proc:

> "/proc is a file system that provides access to the state of each process and lightweight process (lwp) in the system. The name of each entry in the /proc directory is a decimal number corresponding to a process ID. These entries are themselves subdirectories. Access to process state is provided by additional files contained within each subdirectory; the hierarchy is described more completely below. In this document, '/proc file' refers to a non-directory file within the hierarchy rooted at /proc. The owner of each /proc file and subdirectory is determined by the user-ID of the process."

We can use this to our advantage. Assuming we were able to suspend the system and didn't have to do a full shutdown (above), the /proc file system will be intact on our suspect disk. We can first figure out if the process running was part of the original operating system using MD5 and Sun's Fingerprint database (or the database we created using distribution media above). So let's use MD5 to gather up signatures from the running binaries from the /proc file system:

```
mysystem-> md5 /mounted-suspect-disk/proc/*/object/* >proc-md5-sigs
```

This will give us a list of MD5 signatures that we can then examine using the Fingerprint database. This step weeds out binaries that are system binaries and allows us to figure out what was running.

As an example, I take two of the MD5 signatures from processes that were running, and check it against known fingerprints:

```
mysystem-> cat proc-md5-sigs
      [edited to only have a few entries for space]
      MD5 (/proc/22296/object/a.out) = 1a7f2e29e1ee6fb0f5e87d0ba2d8770e
      MD5 (/proc/2332/object/a.out) = 2805a09d9790e70ab0e098f337603656
      MD5 (/proc/2332/object/ufs.32.6.357638) = e725b71635a1c5f1f72eff24cd3e63cc
      MD5 (/proc/2332/object/ufs.32.6.65275) = 84ab84fa8af00324e09627912178c4a0
      MD5 (/proc/2332/object/ufs.32.6.8182) = d60917907d88e9e56251ce0989eebfd4
      MD5 (/proc/2332/object/ufs.32.6.8576) = 422073158f2775bdf119c3847d0d5b64
      MD5 (/proc/2332/object/ufs.32.6.8581) = a11071db878fe24f9e03fb0b53c67bf8
      MD5 (/proc/23418/object/a.out) = 2805a09d9790e70ab0e098f337603656
      MD5 (/proc/9322/object/a.out) = f7f70ef41fdab1b7670582e62270e76c
      MD5 (/proc/23418/object/ufs.32.6.357638) =e725b71635a1c5f1f72eff24cd3e63cc
      MD5 (/proc/23419/object/a.out) = e1ee9f994caeb485767e225f049b3059

mysystem-> sfpC.pl proc-md5-sigs

      1a7f2e29e1ee6fb0f5e87d0ba2d8770e - (/proc/22296/object/a.out) - 0 match(es)

Not found in this database.

      2805a09d9790e70ab0e098f337603656 - (/proc/2332/object/a.out) - 1 match(es)

      canonical-path: /usr/bin/jsh
      package: SUNWcsu
      version: 11.8.0,REV=2000.01.08.18.12
      architecture: sparc
      source: Solaris 8/SPARC
      patch: 109324-01

      e725b71635a1c5f1f72eff24cd3e63cc - (/proc/2332/object/ufs.32.6.357638) - 1 match(es)

      canonical-path: /usr/platform/sun4u/lib/libc_psr.so.1
      package: SUNWkvm
      version: 11.8.0,REV=2000.01.08.18.12
      architecture: sparc.sun4u
      source: Solaris 8/SPARC

      84ab84fa8af00324e09627912178c4a0 - (/proc/2332/object/ufs.32.6.65275) - 1 match(es)

      canonical-path: /usr/lib/locale/en_US.ISO8859-1/en_US.ISO8859-1.so.2
      package: SUNWnamos
      version: 1.0,REV=1999.11.23.15.16
      architecture: sparc
      source: Solaris 8/SPARC

      d60917907d88e9e56251ce0989eebfd4 - (/proc/2332/object/ufs.32.6.8182) - 2 match(es)

      canonical-path: /etc/lib/ld.so.1
      package: SUNWcsr
      version: 11.8.0,REV=2000.01.08.18.12
      architecture: sparc
      source: Solaris 8/SPARC
      patch: 109147-06

      canonical-path: /usr/lib/ld.so.1
      package: SUNWcsu
      version: 11.8.0,REV=2000.01.08.18.12
      architecture: sparc
```

```
        source: Solaris 8/SPARC
        patch: 109147-06

        422073158f2775bdf119c3847d0d5b64 - (/proc/2332/object/ufs.32.6.8576) - 1 match(es)

        canonical-path: /usr/lib/libdl.so.1
        package: SUNWcsl
        version: 11.8.0,REV=2000.01.08.18.12
        architecture: sparc
        source: Solaris 8/SPARC
        patch: 109147-06

        a11071db878fe24f9e03fb0b53c67bf8 - (/proc/2332/object/ufs.32.6.8581) - 1 match(es)

        canonical-path: /usr/lib/libgen.so.1
        package: SUNWcsl
        version: 11.8.0,REV=2000.01.08.18.12
        architecture: sparc
        source: Solaris 8/SPARC

        e1ee9f994caeb485767e225f049b3059 - (/proc/23419/object/a.out) - 1 match(es)

        canonical-path: /usr/dt/bin/dtpad
        package: SUNWdtdst
        version: 1.4,REV=10.1999.12.02
        architecture: sparc
        source: Solaris 8/SPARC
```

Ah! We now know that process /proc/2332/object/a.out was actually /usr/bin/jsh, and we also know that process MD5 (/proc/9322/object/a.out) was not found in the database. This file needs to be examined in more detail. Anything that doesn't match will need additional work to discover what it is. A first check would be to use the what(1) command and then the strings(1) command to see if we can figure out what the file is.

```
  mysystem-> what /mnt-suspect-system/proc/ 9322/object/a.out
        stream.h    1.85   99/12/15 SMI
        isa_defs.h  1.20   99/05/04 SMI
        vnode.h 1.85   99/07/30 SMI
        types.h  1.66   00/02/14 SMI
        feature_tests.h    1.18   99/07/26 SMI
        machtypes.h   1.13   99/05/04 SMI
        int_types.h1.697/08/20 SMI
        select.h 1.16   98/04/27 SMI
        time.h    2.64   99/10/05 SMI
        time.h    1.39   99/08/10 SMI
        time_iso.h  1.199/08/09 SMI
        time_impl.h    1.599/10/05 SMI
        t_lock.h 1.45   98/02/01 SMI
        machlock.h1.21   00/04/27 SMI
        param.h 1.76   00/02/14 SMI
        unistd.h 1.37   98/10/28 SMI
        mutex.h 1.20   98/02/01 SMI
        rwlock.h     1.998/02/18 SMI
        semaphore.h  1.598/02/01 SMI
        condvar.h   1.11   00/03/05 SMI
        cred.h    1.21   97/01/09 SMI
        uio.h 1.29   97/06/29 SMI
        resource.h 1.25   98/06/30 SMI
        seg_enum.h   1.395/12/22 SMI
        poll.h1.28   98/11/23 SMI
        strmdep.h  1.10   98/01/06 SMI
        model.h 1.20    97/09/22 SMI
        strft.h     1.1 99/07/30 SMI
        dlpi.h 1.23     98/04/28 SMI
        bufmod.h    1.9 98/01/06 SMI
        types32.h   1.4 98/02/13 SMI
        stdio.h     1.78    99/12/08 SMI
        stdio_iso.h 1.2 99/10/25 SMI
        va_list.h 1.12  99/05/04 SMI
        stdio_tag.h 1.3 98/04/20 SMI
        stdio_impl.h    1.8 99/06/10 SMI
        ctype.h 1.33    99/08/10 SMI
        ctype_iso.h     1.1 99/08/09 SMI
        string.h 1.24   99/08/10 SMI
        string_iso.h    1.2 99/11/09 SMI
        file.h 1.60     99/08/31 SMI
        stropts.h   1.48    99/08/31 SMI
        conf.h      1.59    99/05/26 SMI
        signal.h 1.54   99/07/26 SMI
        signal_iso.h    1.1 99/08/09 SMI
        siginfo.h   1.54    98/03/27 SMI
        machsig.h 1.15  99/08/15 SMI
        socket.h 1.53   99/11/07 SMI
        netconfig.h     1.20    99/04/27 SMI
        in.h    1.29    00/03/28 SMI
        byteorder.h     1.14    98/04/19 SMI
        un.h    1.9 96/07/12 SMI
        if_dl.h     1.7 98/01/06 SMI
        ioctl.h     1.9 92/07/14 SMI
        if.h    1.23    00/03/28 SMI
        if_arp.h 1.4 98/01/06 SMI
        if_ether.h  1.8 99/03/21 SMI
        in_systm.h 1.5 98/01/06 SMI
        ip.h    1.7 98/08/26 SMI
        udp.h 1.6 99/11/04 SMI
        ip_var.h 1.4 98/01/06 SMI
        udp_var.h   1.2 93/02/04 SMI
        tcp.h 1.14      99/11/04 SMI
        inttypes.h  1.2 98/01/16 SMI
        int_limits.h 1.6 99/08/06 SMI
        int_const.h 1.2 96/07/08 SMI
        int_fmtio.h 1.2 96/07/08 SMI
        ip_icmp.h   1.4 99/04/05 SMI
        netdb.h 1.23    99/12/06 SMI
        inet.h      1.17    99/03/21 SMI
```

This tells us that the binary was compiled using Solaris header files, and it also tells us that this binary probably does some networking functions based on the types of header files it uses, such as tcp.h and socket.h. Let's look at the strings output.

```
mysystem-> strings mnt-suspect-system/proc/ 9322/object/a.out
— TCP/IP LOG — TM: %s —
    PATH: %s(%s) =>
    %s(%s)
    STAT: %s, %d pkts, %d bytes [%s]
    DATA:
    :
    (%d)
```

Well, well. Looks like a network sniffer was running

```
:
(%d)
PKT: (%s %04X)
%s[%s] =>
%s[%s]
Log ended at => %s
%s: alarm
%s: getmsg
%s: alarm finished getmsg() = %i
c6Lqd3Dvn2l3s
(%s)UP?
filtering out smtp connections.
filtering out telnet connections.
filtering out rsh/rlogin connections.
filtering out ftp connections.
Usage: %s [-d x] [-s] [-f] [-l] [-t] [-i interface] [-o file]
-d int set new data limit (128 default)
-s      filter out smtp connections
-f      filter out ftp connections
-l      filter out rlogin/rsh connections
-t      filter out telnet connections
-o <file> output to <file>
Using logical device %s [%s]
Output to %s.%s%s
[Cannot bg with debug on]
Log started at => %s [pid %d]
rlogin
telnet
smtp
DATA LIMIT
TH_FIN
TH_RST
[edited for length]
```

Well, well. Looks like a network sniffer was running. The strings(1) output here matches up to a strings output for the old solsniff.c program that is floating around in cyberspace. This would indicate something really was going on that needs to be investigated.

This last conclusion required a leap of faith because I happen to have seen this output before. You may not be so lucky and may need to look at the binary in more detail, but from just looking at the strings output and the usage line in the suspect binary, I think anyone would agree that this binary was not supposed to be running. The fact that we proved it was not part of the vendor-shipped operating system using MD5 and the Fingerprint database would lead us to conclude, if nothing else, that we did have a problem and that the work we did rebuilding the disk from scratch was justified. From here we need to start looking for how the system was broken into in the first place.

At this point, I think we have exhausted the forensic lite scenario. To take things to the next level and delve into full forensics, we need to examine memory and swap to uncover evidence of how the intrusion unfolded. I'll leave that for a follow-up article on how dissecting swap may uncover exactly how the intrusion occurred.