

## Incident Response Tools For Unix, Part One: System Tools

Holt Sorenson 2003-03-27

This article is the first in a three-part series on tools that are useful during incident response and investigation after a compromise has occurred on a OpenBSD, Linux, or Solaris system. This installment will focus on system tools, the second part will discuss file-system tools, and the concluding article will look at network tools. The information used in these articles is based on [OpenBSD 3.2](#), [Debian GNU/Linux 3.0 \(woody\)](#), [RedHat 8.0 \(psyche\)](#), and [Solaris 9](#) (aka Solaris 2.9 or SunOS 5.9).

Depending on what operating system you're using, some of the software discussed in this article may not be part of the base install. In this case, the reference section at the end of this article provides several ways to acquire the tools in question [\[1\]](#).

### The Soapbox

The best tools that can be utilized in response to the intrusion threat are not ones that will be discussed in detail in this series. The tools that will be covered are discussed in the context of triage after an intrusion has happened. Intrusions are generally preventable. Using techniques such as keeping systems up-to-date by patching vulnerabilities, configuring a system with the minimum required services, using operating systems that are designed with security in mind, and using kernel patches that harden the system go a long way toward making sure you never have to use the tools discussed in this series.

Readers should also understand that once an attacker has had control of a system, it's nearly impossible to trust the system again. Many of the tools we will discuss operate in user-land. A rogue kernel module could present a healthy looking system, when, in fact, the system has been compromised. The rogue module can hide itself in such a way as to be undetectable while the system is running because of the current design of operating systems, unless countermeasures [\[2\]](#) are used. This means that there are times that you can't trust the output of the tools that you're using. Being skeptical, pedantic, and meticulous is key when responding to an incident.

The tools you use to analyze a system should be tools you can trust to not have been modified. There are several technologies discussed below [\[3\]](#) that provide tools that are stored on offline read-only media that are likely to be more trustworthy than binaries on a live system that has been compromised.

You've done all the right things, you're obsessive about following best practices, and your machine still got compromised. So what now?

### Breaking Out the Toolbelt

Let's start by examining the tools that we'll use later to respond to a call about a possible security incident. I try avoid discussing information that **man(1)** can provide, especially since command arguments (also known as flags, options, or parameters) can differ from one operating system to another. Exploring such information is left as an exercise to the reader. We all like calisthenics, right?

**vmstat** is a program that allows one to get a quick look at statistics about the memory, CPU, and disk subsystems. **vmstat** is generally executed for a short period of time so that one can see a trend in the utilization of the subsystem one is interested in. Often **vmstat** helps one know where to dig deeper when a system is not performing as expected.

**mpstat** is a program that is available on Solaris and Linux that allows one to see statistics about processor utilization. **mpstat** provides an option that allows one to display statistics for a specific CPU on multiprocessor systems. **vmstat** doesn't have this capability.

**iostat** displays statistics that are more detailed than those displayed by **vmstat** about the disk subsystem.

**sar**, **sa**, **lastcomm**, and **last** all allow one to examine historical data as well as more recent events on the system. **sar** is a system performance analysis tool available on Solaris and Linux. The performance data one can examine is similar to that displayed by **vmstat**, **mpstat**, and **iostat**. The data **sar** uses is saved over time, so that one can look into the past. **lastcomm** is a tool that displays commands that have been run on the system, starting with those most recently executed. It uses data contained in the process accounting system to accomplish this. **sa**, available on \*BSD and Linux, gives users more options in displaying data gathered by the process accounting system.

**ps**, which stands for "process status", is used to show processes executing on the system and information about them. A simplified definition of a process is that it is a unit of execution that has its own memory space. Typically a program that is running has one or more processes associated with it.

**top** displays information similar to the **ps** command, but **top** orders the commands by CPU consumption. It also updates the display after a user specified period of time that defaults to two seconds in most cases.

**lsof**, or list open files, shows all the files that the operating system currently has open. Unix operating systems consider almost everything to be a file, so **lsof** shows a significant amount of information about the state of the operating system.

The **file** command uses so-called magic numbers to determine what the file is. Different file formats begin with certain hex values that indicate the contents of the file.

**readelf** displays in detail the ELF (Executable Linking and Format) headers of a binary file. These details can help in determine what functions the executable performs.

**od** outputs a dump of file contents in various user specifiable formats. **od** is useful for looking at the raw bits in files with some interpretation on their contents.

**ldd** reads the contents of the ELF headers that show what shared object libraries the executable depends on.

**strings** shows ASCII strings at least four characters or longer (this is user specifiable, however) in a file. This is useful for finding user readable strings that might indicate the purpose of the binary.

**find** is a command that recursively searches for objects on the file-system from a user specified point. It can be useful for finding objects (usually files or directories) that match user-specified criteria.

**strace** is a utility that starts a process or attaches to a currently running process and shows all the system calls that the process is making. This can be useful for classifying the runtime behavior of a program to determine what the purpose of a program. **strace** is available on Linux. **strace**'s analogue on Solaris is **truss**. \*BSD employs a program called that performs a similar role called **ktrace**.

**sudo** enables administrators to give users the ability to run commands as users other than their own without giving the user root's password.

**grep** or global regular expression print, searches a pattern space (typically a file) for a user specified pattern. **grep** uses [regular expressions](#) -- a means of defining search criteria -- that are compared against strings in the pattern space. Lines that contain a match are printed to std-out (generally your terminal, aka "the screen"). GNU **grep** has extensions to your typical **grep** that make it more powerful. For Perl junkies, [pcregrep](#) allows one to specify the regular expression in Perl's regular expression dialect.

**less** is a pager, or a program that displays pages of text, one screenful at a time. **less** is more than **more**, so spending time getting acquainted with its features are recommended because it has a more robust feature-set. In other words, you should ditch **more** for **less**.

A fairly comprehensive list of sites that contain information on how to compile these tools together on CD-Roms for various operating systems is available in the reference section of this article entitled "CD-ROM and Floppy distributions"[\[3\]](#) . Now you're set to investigate the running system and see what's going on.

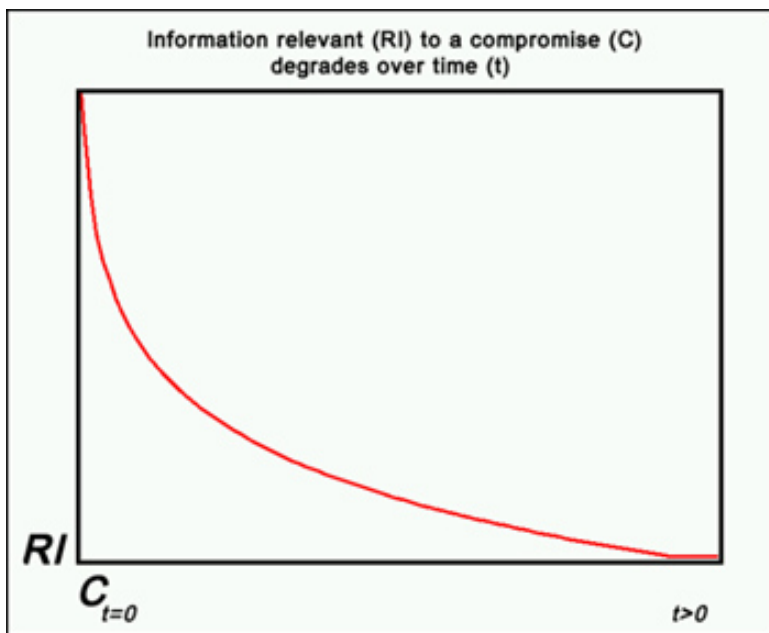
### **Luck is When Preparation Meets Opportunity**

Unfortunately, luck, like technology, doesn't discriminate. Many things came together for your attacker to compromise your OS. The vulnerability that was used to compromise your system was created by the application or OS vendor. The vulnerability hasn't yet been announced to the security industry at large, or the vendor itself. Your attacker was on IRC bragging about a system that had been cracked and traded access to the system for this zero-day 'sploit that was used to compromise your system. |<-r4D !\_337 for the cracker, unfortunate for you. As a security administrator, your game has to be perfect at all times. The cracker only has to get it right once.

However, you've done some preparation work. You have an incident response policy[\[4\]](#). You have a CD-ROM or floppy distribution[\[3\]](#) that has tools that allow you to check the operating system without having to trust the binaries on the machine since it has been compromised. If you're using floppy disks, they have the write protect on, of course. You've extensively studied excellent resources on forensics[\[5\]](#).

You jump on the machine and go to work. You grab your incident response notebook and log the date and time. You begin a copious, even obsessive, note-taking frenzy that documents the time and details of each step you take. You mount the CD-ROM or floppy and use software on it to record and store the pertinent bits in a safe and secure place keeping in mind the order of volatility [\[6\]](#).

The reason the order of volatility is important is because the quality and quantity of relevant information after a compromise has occurred decreases quickly. Sagacious attackers do their best to steepen the curve at which this degradation occurs.



### Getting to Know You, Getting to Know All About You

Prior to this intrusion, you've spent a lot of time with your system. You know it inside and out. The system is well documented. It has current backups. You have process accounting [7] (or system accounting if you're using \*BSD) turned on. You use the system activity data collector (sadc)[8] to save performance data to the system. You manage your logs, but have enough disk space to not have to rotate them too frequently. You send syslog data to a secure machine that is hardened against attack.

You get a call from one of your Webmasters who notices that the CPU is busy on a Web server that usually only serves a few thousand hits per day. The Webmaster is concerned that something is awry.

**vmstat** and **mpstat** (not available on \*BSD) show that is the CPU is being consumed by one or more processes in user-space, but that the memory and i/o subsystems are not significantly utilized. **iostat** also shows that the disk subsystem is bored.

```

$ vmstat 1 4
   procs           memory      swap             io           system           cpu
  r  b  w   swpd   free   buff  cache   si  so   bi   bo   in   cs  us  sy  id
  1  0  0    376   7756 29772 570960    0  0    7    3  441   397  87   5   8
  5  0  0    376   6728 29772 570960    0  0    0    0  498  1249 100   0   0
  6  0  0    376   7240 29772 570960    0  0    0    0  652  1563  97   3   0
  6  0  0    376   7604 29772 570960    0  0    0    0  536  1323  97   3   0
$ mpstat 1 4
20:51:21      CPU   %user   %nice %system   %idle   intr/s
20:51:22     all 100.00    0.00    0.00    0.00    479.00
20:51:23     all 100.00    0.00    0.00    0.00    496.00
20:51:24     all 100.00    0.00    0.00    0.00    499.00
20:51:25     all  97.00    0.00    3.00    0.00    481.00
Average:     all  98.00    0.60    1.40    0.00    486.60
$ iostat -dk 1 4
Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
dev3-0              0.00         0.00         0.01         73         1296
dev3-0              0.00         0.00         0.00         0           0
dev3-0              0.00         0.00         0.00         0           0
dev3-0              0.00         0.00         0.00         0           0

```

**sar** (sa on \*BSD) shows that the cpu started being used the previous night at 03:17. **lastcomm** shows that the FTP client was run by root several times in the minutes preceding 03:17. **last**, which shows recent logins, doesn't show when or where root logged in from this time. Additionally, your site uses **sudo** to manage root privileges. There are two sysadmins that could theoretically login as root, but they don't in practice, especially at 3 AM. At this point, you decide to get out your CD of binaries because it looks like this rabbit hole could run deeper. Now you run **top -d1** (that's the number one, not the letter "l") and see that a process named apache is using up 100% of the CPU.

You search apache's error\_log using GNU **grep** because of the most excellent options, -A, -B, and -C that let you specify the number of lines you want to see after, before, or around a matched line. GNU grep also lets you specify -E for extended regular expressions. You also check the system logs. You don't find anything in the logs that seems odd.

You continue your search by running **ps -efcyl** (Sol9) **ps -efcym --headers** (Deb 3.0, RH 8.0) or **ps auwxhkwvl** (OBSD 3.2). You find the apache process running, but this time something strikes you as odd. There is only one apache process and there are multiple httpd processes. Then you remember that httpd is actually the Apache server because httpd is the binary that gets executed in the apachectl file and you are running Apache with the pre-fork MPM, so even if the binary had been renamed to "apache" there *should* be multiple processes running. Hmm... something smells fishy...

You use **lsof -p <pid> [9]** and find that the supposed apache process has a file called john.pot open. A quick

Google search suggests that the likely culprit is a password cracker called "John the Ripper", which would explain the CPU being taxed.

You decide to check out this so-called apache binary. You run **file apache** and **file** says it's a ELF executable. You decide to dig deeper and run **readelf -a apache**. **readelf** corroborates **file's** assertion that this is an ELF executable. **od -xc apache|less** also shows the ELF magic number (7f454c46) at the beginning of the file. **ldd** shows that the shared object library files that the apache binary links to is a significantly smaller list than your real httpd binary links to:

```
$ ldd ./apache
    libc.so.6 => /lib/libc.so.6 (0x4001f000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
$ ldd /usr/sbin/httpd
    libm.so.6 => /lib/libm.so.6 (0x7002c000)
    libcrypt.so.1 => /lib/libcrypt.so.1 (0x700c4000)
    libdb.so.2 => /lib/libdb.so.2 (0x70100000)
    libdb2.so.2 => /lib/libdb2.so.2 (0x70120000)
    libexpat.so.1 => /usr/lib/libexpat.so.1 (0x70180000)
    libdl.so.2 => /lib/libdl.so.2 (0x701b4000)
    libc.so.6 => /lib/libc.so.6 (0x701c8000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x70000000)
```

You run **strings | grep -i john** and the following is displayed:

```
$ strings apache | grep -i john
/usr/share/john/password.lst
/etc/john.ini
~/john.pot
/etc/john.ini
john
John the Ripper Version 1.6 Copyright (c) 1996-98 by Solar Designer
/etc/john.ini
/etc/john.ini
/etc/john.ini
```

You decide to watch what the **apache** process is doing using **strace -fp `pgrep apache`** (Deb 3.0, RH 8.0), **truss -fp `pgrep apache`** (Sol9), or **ktrace -dip <pid>**(OSD 3.2).

```
# strace -fp `pgrep apache`
--- SIGALRM (Alarm clock) ---
sigreturn()                = ? (mask now [])
--- SIGALRM (Alarm clock) ---
sigreturn()                = ? (mask now [])
--- SIGALRM (Alarm clock) ---
sigreturn()                = ? (mask now [])
_llseek(4, 65536, [65536], SEEK_SET) = 0
read(4, "\2YUz\0\2Z\0241\0\2ZAldkmnr\0\2ZDa\0\2ZElnrd"... , 4096) = 4096
read(4, "\2\5Ma\0\2\5Te\0\2\6\n(\0\2\6>%\0\2\10\n&\0\2\10\0210\0"... , 4096) = 4096
read(4, "IPVm\0\2(%L\0\2((H\0\2(/CMRBNTUWcr\0\2(5"... , 4096) = 4096
read(4, "fgq\0\2BCh26o\0\2BDaikmoy\0\2BEanreltc"... , 4096) = 4096
read(4, "ag\0\2N\231\0\2N\25j\0\2N\0269\0\2N\30f\0\2N!ds\0\2N)"... , 4096) = 4096
read(4, "\0\2_9LRMTes\0\2_:6Z\0\2_>%\0\2_?)Cw\0\2_"... , 4096) = 4096
read(4, "%Antv\0\2%Cr\0\2%Dd\0\2%Gg\0\2%Ke\0\2%Lls\0"... , 4096) = 4096
read(4, "rtlpbdgiuv\0\2GBeam\0\2GDuy\0\2GEnert\0"... , 4096) = 4096
read(4, "We6hilw\0\2RYasdpcgilmno\0\2S\0205c\0\2S\21"... , 4096) = 4096
read(4, "68jDxMEBTIRNLAGSKCPOqVHFUZWJ$%#^"... , 4096) = 4096
read(4, "\0022In\0\0022Ke\0\0022LiEe\0\0022Ma\0\0022NEe\0\0022P"... , 4096) = 4096
read(4, "I5y\0\2I6e\0\2I7ae\0\2IANmrs9t67dhlpuz"... , 4096) = 4096
read(4, "Eprylsenbu9ak013ft!Lcox\0\2SFi9aey"... , 4096) = 4096
read(4, "o149dlytwMbckr0u\0\2_Xixae01537926"... , 4096) = 4096
read(4, "2\0\2-!NCLS\0\2-%RS\0\2-)ETDNS\0\2--M\0\2-"... , 4096) = 4096
read(4, "!*bfgk\0\2GJu\0\2GLEiayYs\0\2GMAoegu\0"... , 4096) = 4096
read(4, "Utsdbgmpy\0\2PWa\0\2PXy\0\2PY1!27sy.05"... , 4096) = 4096
read(4, "w!9bAjp5clgkr068EFSYf$.?CJKLW]n\0"... , 4096) = 4096
--- SIGALRM (Alarm clock) ---
sigreturn()                = ? (mask now [])
```

You use **grep -n** to poke around in the source code of [John the Ripper](#) and find that there is a function called `sig_timer_emu_tick()` in a function called `crk_password_loop()`. You look at the `sig_timer_emu_tick()` and find that it raises `SIGALRM` after `timer_emu_max` is reached. This explains the regular `SIGALRMs` that **strace** is showing. You use **ls -p `pgrep apache`** to see what file is associated with file descriptor (fd) 4 that `read` is getting data from and find that fd 4 is associated with a file called `all.chr`. You look in the `john.ini` file that's included with the source and find that it references `all.chr` in the incremental modes section.

Given all this evidence, you feel certain that this is indeed the password cracker, John the Ripper. Since neither you nor any of your colleagues installed this program, it's beginning to look like the Webserver has been compromised.

## Conclusion

In the next article, we'll continue to investigate what has happened to this Webserver using file system tools.

Remember, the best preparation for any incident threat is prevention. When a compromise does occur, the more work you've done to know your systems, have policies in place, and practice incident response techniques, the higher your odds are for successfully containing and eradicating a threat that has materialized.

## References

### [1] Acquiring/Installing software

- [OpenBSD ports system](#)
- [FreeBSD ports system](#)
- [Freeware for Solaris](#)
- [HOWTO install packages on a Debian system](#)
- [Installing packages on a RedHat system](#)
- [GNU binutils \(readelf\)](#)
- [lsof](#)
- [top](#)
- [sudo](#)

### [2] Kernel Modules Countermeasures

- [Dino Dai Zovi's paper on kernel rootkits](#)
- [Sealing the linux kernel](#)
- [gr-security linux kernel patch](#)
- [LIDS linux kernel patch](#)

### [3] CD-ROM and Floppy distributions

- Here is a link that explains how to make CD-ROMs on Linux: <http://www.tldp.org/HOWTO/CD-Writing-HOWTO.html>
- [F.I.R.E. \(Forensic and Incident Response Environment\) Bootable CD](#)
- [Knoppix Linux](#)
- [How to Make a Bootable, Full System OpenBSD CD-ROM](#)
- [LNX-BBC mini-linux distribution](#)
- [Google's Tiny Linux distributions list](#)

### [4] Incident Response Policy

- [Handbook for Computer Security Incident Response Teams](#)
- [Expectations for Computer Security Incident Response](#)
- [Recommended Internet Service Provider Security Services and Procedures](#)
- [Recovering from an Incident \(CERT\)](#)
- [Security Practices and Implementations \(CERT\)](#)
-

## [5] Forensics

- [Incident response and Forensics Resources](#)
- [Forensics resources maintained by Dan Farmer](#)
- [Computer Forensics Resource Center](#)
- 

## [6] Order of Volatility

- [Computer Forensics Analysis Class: Introduction, Pg. 14 \(Dan Farmer & Wietse Venema\)](#)

## [7] Process accounting

- [Solaris 9 SysAdmin Guide: Resource Management and Network Services, Ch 7](#)
- [Enabling Process Accounting on Linux HOWTO](#)
- [OpenBSD sa\(8\) man page](#)

## [8] System Activity Data Collector (sadc)

- [sadc\(8\) Linux man page](#)
- [Solaris 9 SysAdmin Guide: Monitoring System Performance \(Tasks\), Ch 24](#)

## [9] Isof alternatives

Don't have **isof**? These techniques might help.

Solaris 9:

1. use **ps** to find the pid of the process you're interested in.
2. run **/usr/proc/bin/pfiles** .
3. Look for the string 'ino:' in the output. The number that follows is the 'inode number'.
4. run **find -inum <inode number>**

OpenBSD 3.2:

1. run **fstat -v**.
2. Look in the INUM column to find the the inode numbers that a particular process has open.
3. run **find -inum <inode number>**

Poking around in /proc (Deb 3.0, RH 8.0):

1. **ls -l /proc/<pid>/fd**

## Further Study

The following commands aren't covered in this series, but can be helpful for analyzing systems or the binaries on them: **nm**, **size**, **ar**, **objdump**, **what** (OS/390 3.2), **ltrace** (Deb 3.0, RH 8.0), `/usr/proc/bin/*` (Solaris), **w**, **who**, **and fuser**. Becoming familiar with them is encouraged.

Most Unix systems have an on-line reference manual called **man**. Spending some time acquainting oneself with the manual is quite useful to help one understand the commands discussed above. To access information for a command, one types **man <command>**. If the command one was interested in was **vmstat**, one would type **man vmstat**. Depending on the operating system you use and its set-up, there are several other commands that can help one understand commands and concepts related to Unix systems. **whatis** searches the command field of the NAME section of the man page for whatever string you pass as an argument. **apropos** searches the whole NAME section of the man page for whatever string you pass as an argument. **man -k** also provides similar, and in some cases, the same functionality. The GNU **info** command is a documentation system that acts as documentation for most GNU software. If you have questions related to Perl, the **perldoc** command is the weapon of choice. **whatis**, **apropos**, **info**, and **perldoc** are all invoked the same way **man** is.

*Holt Sorenson is currently employed by [Counterpane Internet Security](#) where he takes care of network and computer security. He's not as obsessive about silicon-based beasts as he used to be, as there are several instances of bio-ware that have come along that are much more interesting. He'd like to thank the following, ordered by surname for their help in reviewing this article: Ben Laurie, Chris Odhner, Bruce Potter, and the anonymous others. Comments on the article are appreciated and can be emailed to [Holt](#).*

[Privacy Statement](#)

Copyright 2006, SecurityFocus