SHADOW Version 1.7
Installation Manual

William D. Ralph
<RalphWD@nswc.navy.mil>

21 September 2001

Abstract

SHADOW is an Intrusion Detection System built on inexpensive hardware and free software. This document details how to build, install and configure the system.

Contents

1 Introduction

SHADOW is an Intrusion Detection system based on inexpensive PC hardware running Open Source, public domain, or freely available software components. A SHADOW system consists of at least two pieces: a sensor located at a point near an organization's firewall; and an analyzer located inside the firewall. SHADOW performs traffic analysis; the sensor collects address information from all IP packets that travel between an organization and the Internet; the analyzer examines the collected data and displays user defined "interesting" events on a web page. SHADOW is based on *tcpdump* and *libpcap* software packages developed at the Lawrence Berkeley Laboratory to collect the packets and to filter the collected traffic according to user defined criteria. Using Perl software developed at the Naval Surface Warfare Center Dahlgren Division, the filtered results are displayed via an Apache web server on an analyzer station being regularly monitored by an intrusion detection analyst. The key to effective use of SHADOW is intelligent definition of the *tcpdump* filters based on the network environment and educated recognition of known and potential exploits from traffic patterns.

This document outlines the steps for installing a SHADOW system. These steps are very broad and leave a lot of work for the installer. The person performing the installation must be intimately familiar with the workings of his Linux system and the process of installing open source software. If you don't know Linux thoroughly, you are going to have trouble!! It's possible to install a working SHADOW system, but it will be tough. Steps not fully documented here can be show stoppers; for example, if you don't have a C compiler on your system, you cannot install any software from the source, and you will have to find a way around that problem. Another example: the SHADOW scripts are written in Perl. If your system doesn't have Perl, you have another problem.

This document concentrates on installation for a Red Hat Linux system. That is the operating systems used at NSWC to develop SHADOW and to run it in "production" mode. SHADOW is known to work on other systems, however, the procedures outlined here have been developed for and tested only on Red Hat Linux. You will have to adapt them to your particular flavor of Unix or Linux.

2 History

Before Intrusion Detection became popular, before hackers made the evening news, analysts at research and development sites were interested in examining the network traffic traversing through their sites. The Naval Surface Warfare Center/Dahlgren Division is one such site. But since "computer security" had not yet made it to the mainstream of consciousness, little

money was available for buying or building custom devices to examine patterns of traffic. So, Stephen Northcutt, at NSWC, began building sniffers from old discarded workstations and public domain software. He used a software package from Texas A & M University called "*netlog*," which consists of three programs, *icmplogger, tcplogger*, and *udplogger*. These programs ran only on Sun equipment collecting IP packets of the appropriate type. Auxiliary programs read the collected data and displayed it in human readable format. Stephen wrote a collection of Perl scripts to parse the data and create web pages on which were written "events of interest" that his scripts found in the raw data.

At that point, SHADOW was born. It was not yet named, and it had a lot of growing up to do. Unfortunately, using old Sun boxes, proved to be limiting. They were functional but obsolete in terms of computing power. They ran *netlog* which provided the data to generate the web interface. But as the network traffic increased, and the sources of Suns started drying up, alternatives were sought. NSWC discovered that performance could be improved by using PCs and Linux instead of the recycled Suns. The viable lifetimes of new PCs were longer than the Suns. Unfortunately, the *netlog* package was not available for Linux on the Intel x86 hardware. A package called *tcpdump* from Lawrence Berkeley Laboratory, was available which offered more capability than *netlog*. The switch to *tcpdump* was made, and the project was named SHADOW. SHADOW was originally built on Red Hat Linux Version 5.0. It has been migrated to every subsequent version of Red Hat Linux.

3 Open Source Software

In the days of the "big iron," those large scientific computer mainframes like Control Data 6600s and Crays, government sites never fully depended on the computer vendors to supply and support the operating system software. In fact, when the first Cray machines were built, Department of Energy sites had a time-sharing operating system ready to run on the hardware before the vendor did. There were and still are certainly advantages to this approach. Operation of the site computers was not dependent on the vendor's priorities or abilities. Security, which was a high priority of the government organization but not necessarily of the vendor, could be built in from the start. If a problem occurred with the system software, the source code and usually the developers of it were available to assist in finding a solution. Given the availability of the source code, (and someone technically astute enough to find and fix it), any software problem can be solved; without it, you are at the mercy of the vendor.

From a security perspective, the only way to insure that a piece of software does what it's supposed to do and only what it's supposed to do is to examine the code. Commercial software companies will gladly sell you their code in binary form and claim that it will

function as advertised. But read the license. It says that the software may not do what it is supposed to do; it may have unpredictable, or even damaging, side effects on your hardware or other software; but the company is not responsible. Once you accept the license, the risk is yours alone. That says a lot about what such companies feel about quality and responsibility. Caveat emptor.

4 Installation

Installation of SHADOW will be described in two sections, one for the sensor and one for the analyzer.

4.1 How to Build a SHADOW Sensor.

4.1.1 Hardware Requirements.

A SHADOW sensor just reads data packets from a network interface and writes those packets to disk, so there is no need for an expensive video card or sound card. It runs well on 128 MB of memory. At NSWC, so far, a 350 MHz Pentium II has never missed a packet on a 10/100 mbs network, so anything faster than that should likewise suffice. If your network is running gigabit speeds, you'll have to use the trial and error approach to obtaining a sufficiently fast sensor. A SCSI card and SCSI disks are highly recommended because they are faster and more expandable than the IDE drives normally available on PC class machines. Nothing smaller than a 9 GB disk will hold your raw files for long, so at least an 18 GB Ultra-160 SCSI drive would be preferred. To make the sensor externally invisible, you should use two network cards; one will have now IP address, and will collect the raw data on the principal pathway into your network, and the other will be logically placed inside the site firewall for communication with the SHADOW analyzer, also inside the firewall.

4.1.2 Install and Patch the Sensor's Operating System Distribution.

This is not going to be a step-by-step guide through the installation of the Red Hat Linux operating system distribution. The official Red Hat Installation guide, [RHL-Install] will suffice for that purpose. This document will enumerate some points at which a divergence from a "normal" installation step, or a specific configuration choice should be made.

The first choice to be made is the "class" of installation. Red Hat offers five choices: workstation, server, laptop, custom, or upgrade. If you are upgrading an existing SHADOW sensor

to a later version of the operating system, choose "upgrade." Otherwise, choose "custom." The sensor is basically a limited usage computer. It doesn't need a lot of the functionality that Red Hat would have you install for its workstation, server, or laptop classes. Using the custom class will enable you to pick and choose the packages to be installed.

Shortly after beginning the install, you have the opportunity to partition your disks. At NSWC, only 68 bytes of the IP packet are collected. Still, a gzipped file of one hour's raw output can exceed 300 MB during the busiest times of day. For most weekdays, the daily total of gzipped hourly files can exceed 2 GBytes. That's a lot of data, so you need to maximize the size of the partition on which the data will reside. You can make the root partition about 4 GB and allocate the rest of your 9 GB or larger drive for a "/LOG" partition to hold the raw data files. If you expect your traffic volume to be significantly different, adjust the disk size and or partitions accordingly.

Later the installation process presents the network configuration window. If you have installed two network interface cards (NICs), you will be offered the opportunity to configure each of them. Be sure to configure only one of them with the IP address, netmask, etc. that the installation process asks for. Remember, one NIC of the sensor will be made "invisible;" it has no assigned address and will not be visible to other machines on its network segment.

The "Firewall Configuration" section of the installation process is intended to provide some kernel level capability to filter, block, and/or log incoming IP packets. The same result is achieved by manually constructing an *iptables* script, as described in 4.1.9 and C, so you may bypass this installation step, if you wish. If not, choose the "High" security level. The sensors need to be protected as much as possible. But remember, for SHADOW, the analyzer must be able to connect to the sensor to fetch the data files and for periodic maintenance. Detailed configuration of SSH will be described in 4.2.7 below.

Eventually, the installation process will reach the Package Group Selection section. The sensor needs no X Windows, Multimedia, Games, Sound Services, Development Tools, etc., etc. If you leave a lot of the tools on the system, you are providing those tools to potential intruders. Granted that a sophisticated intruder may only be inconvenienced by the lack of those tools, any inconvenience foisted upon such folks may give you extra time to detect and react to a system compromise. Use some common sense about what packages to install. If it is not directly related to the job of a SHADOW sensor, i.e. reading and storing network packets, it's not needed. If it is needed later, it can be installed then. Since you should build a custom kernel for the sensor, (see 4.1.3), you may want to leave the development tools until after that kernel rebuild is done. You can however, build a custom kernel on your analyzer and remotely install it at a later time.

After the distribution installation has completed and the system has rebooted, go to Red Hat Updates `http://updates.redhat.rpm/` and download the updates for the packages in RPM format for your distribution. These are the "Patches" to the distribution that have

been made since the it was released. ***IT IS VERY IMPORTANT TO INSTALL THESE PATCHES AND TO PERIODICALLY MONITOR THE SITE FOR ADDITIONAL ONES!!!!*** The security risks are constantly changing, if you assume that you are up to date with your security preparedness, someone out there is working diligently to prove you wrong. Once you've collected all the RPMs (Red Hat Package Manager, the files in which Red Hat furnishes its software), into a directory, use the *rpm* command to update your distribution:

```
rpm -Fvh *.rpm
```

Note that you should not fetch and apply updated RPMs for the kernel. The kernel will be built and customized to your hardware in step 4.1.3 below.

If you installed two NIC cards, add the line "*ifconfig $PASSIVE_IF up*" to the */etc/rc.d/rc.local* file, where *$PASSIVE_IF* is replaced by the name of the inteface that will be invisible. This will insure that at each system boot, that interface will be brought up with no assigned address so that *tcpdump* can capture the packets "invisibly." This makes the sensor difficult to detect and attack, since with no address, it cannot be the destination of any packets.

4.1.3 Build a Custom Sensor Kernel.

As released, the Red Hat operating system includes a kernel that is configured to recognize and handle a large variety of hardware configurations. It is built with kernel loadable modules enabled; small pieces of code that the kernel can load and execute as needed. For example, if your machine has a 3Com Ethernet card, the boot process will detect it and load the appropriate 3Com module when it brings up networking. Any other Ethernet loadable modules will be ignored. For maximum flexibility in the installation process, Red Hat builds a large number of kernel loadable modules for many of the most likely hardware configurations. This greatly simplifies initial OS installation. As the installation process proceeds, hardware components are recognized, and the kernel loads the appropriate driver to interface to that component. The kernel is able to dynamically reconfigure itself to match the hardware it finds.

Of course, there are counterpoints to this flexibility. Many modules that are never needed are built and indexed for hardware configurations that will never exist on your system. The kernel itself is configured to use only those instructions common to all x86 Intel processors, i.e. the 80386. Extended instructions for later processors, e.g. the Pentium, Pentium II, or Pentium III are not available unless the kernel core is recompiled for a specific processor. Kernel loadable modules may also present potential security concerns. It has been documented that the creation of a "trojan" kernel module is possible. So, for example, someone creates a module to give a particular process special privileges if the module is loaded into

memory by the kernel. If this module is given the name of the your system's sound card, the kernel will load it when the sound card is needed. When that happens, the creator of the module can operate with kernel privileges.

To avoid those problems and to optimize the performance of the Linux kernel for your specific hardware, build a custom kernel. The latest version of the Linux kernel should be downloaded from Kernel.org `www.kernel.org` to insure that it has the latest fixes and capabilities.

Here are the steps to build a custom kernel:

1. *Download the latest kernel, e.g. linux-2.4.9.tar.gz.*
2. *rm /usr/src/linux*
3. *cd /usr/src; tar xvfz /some/path/linux-2.4.9.tar.gz*
4. *ln -s linux-2.4.9 linux*
5. *cd /usr/src/linux*
6. *make menuconfig(or xconfig if running X Windows)*

At this point, a menu will appear that presents you with all the possible hardware and software configurations. Answer 'N' to all hardware not on your system. Answer 'N' to all software unnecessary for your sensor, e.g. sound card, ISDN, etc. Answer 'N' to "Enable loadable module support." When you complete your configuration, at the end of the menu, click on Exit. which then creates a file, ".config" which contains all the configuration options you selected during the process. See Appendix A for a copy of a sensor *.config* file that defines the kernel configuration for a sensor at NSWC.

Once your configuration file matches your hardware and desired software, build the new kernel and put it into place by:

1. *cd /usr/src/linux*
2. *make dep*
3. *make bzImage*
4. *make modules*
5. *make modules_install*
6. *cp System.map /boot/System.map-{$VERSION}*
7. *cp arch/i386/boot/bzImage /boot/vmlinuz-{$VERSION}*
8. Edit the file */etc/lilo.conf* to include the new kernel you just built.
9. Run */sbin/lilo* to re-install the boot loader with the new kernel information.
10. Re-boot your system to see if the new kernel will run.

    Where $VERSION is the version of the kernel you are building, e.g. 2.4.9.

Be sure to study the document on how to build a Linux kernel, which can be found at: Kernel-HOWTO `http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html`.

4.1.4 Unpack the SHADOW-1.7 Software.

The SHADOW software comes in a gzipped tarball: SHADOW-1.7.tar.gz. Choose a location for unpacking the software. "*/usr/local/SHADOW*" might be a good place, so you can unpack the tarball in that location:

1. `mkdir -p /usr/local/SHADOW`
2. `cd /usr/local/SHADOW`
3. `tar xvfz /tmp/SHADOW-1.7.tar.gz`

4.1.5 Install *Tcpdump*.

Unfortunately, with release 6.2 and later of Red Hat Linux, Red Hat decided to abandon the *tcpdump* that was "standard" from Lawrence Berkeley Labs. They applied a number of patches that extended the capabilities of *tcpdum*p. Those patches also changed the format of the output. SHADOW works by parsing the output lines from *tcpdump*, so it fails with the new Red Hat version. When the tcpdump.org `www.tcpdump.org` site assumed development and maintenance of *tcpdump*, the version that they build was compatible with what SHADOW expected. However, some bugs have been found in *tcpdump* and fixed at NSWC. The fixes and findings have been sent to tcpdump.org, but, given their lack of response, there was no alternative but to furnish a version of *tcpdump* with the SHADOW package. Alternatives to *tcpdump* for subsequent SHADOW releases are being examined.

Install *tcpdump* with the following command:

```
cd /usr/local/SHADOW/accessories/rpms
rpm -Uvh --force tcpdump-3.6.2-wdr2.i386.rpm
```

The source, patches, and spec files are also furnished, so non-Red Hat Linux users can compile and install *tcpdump* as well.

4.1.6 Install OpenSSH.

The analyzer must communicate with the sensor. The mechanism chosen to implement that communication is *SSH*, the secure shell. The sensor may be located outside the site firewall, and thus may be more vulnerable to compromise or evesdropping. *SSH* provides a mechanism for communication between the analyzer and sensor that enhances authentication and is resistant to evesdropping. Because of proprietary code, patent, and copyright restrictions

on the original *SSH*, *OpenSSH* was selected for implementation with SHADOW. Red Hat Linux 7.1 furnishes version 2.5.2p2 *of OpenSSH* in its distribution in five RPMs: *openssh, openssh-server, openssh-clients, openssh-askpass,* and *openssh-askpass-gnome.* Included in the SHADOW package is *OpenSSH* version 2.9.9p1 in a single RPM. Install it by:

```
cd /usr/local/SHADOW/accessories/rpms
rpm -Uvh --force /openssh-2.9.9p1-wdr1.i386.rpm
```

The tarball for *OpenSSH* 2.9.9p1 is included as well .

Since the sensor will be the recipient of *OpenSSH* connections from the analyzer, the *sshd* daemon must be configured to accept and authenticate those connections. Here is a list of the changes to the default */etc/ssh/sshd_ config* file, preceded by "->":

```
#       $OpenBSD: sshd_config,v 1.34 2001/02/24 10:37:26 deraadt Exp $

# This sshd was compiled with PATH=/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin

# This is the sshd server system-wide configuration file.   See sshd(8)
# for more information.

Port 22
->Protocol 2,1
.   .   .
#
# Don't read ~/.rhosts and ~/.shosts files
->IgnoreRhosts no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
->IgnoreUserKnownHosts yes
->StrictModes yes
->X11Forwarding yes
.   .   .
->RhostsAuthentication yes
#
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
->RhostsRSAAuthentication yes
#
->RSAAuthentication yes
```

For this to work correctly, you must insure that the sensor file */root/.ssh/authorized_ keys2* contains the SSH public keys of the SHADOW user generated on the analyzer. Detailed instructions about installing SSH between the analyzer and the sensors are provided in the section 4.2.6. The book cited in Ref. [SSH] is highly recommended; it's one of the best available references for understanding SSH's workings and how to configure it. Configuring *OpenSSH* is not particularly difficult, once you've done it a few hundred times.

4.1.7 Configure SHADOW.

After the software package is unpacked, it needs to be configured. Three files require customization: *sensor_init.sh, std.ph,* and *std.filter.*

Here are the requirements:

1. For *sensor_init.sh:*
   (a) Set the variable SENSOR_PATH to point to where you've installed the SHADOW package, e.g. */usr/local/SHADOW/sensor.*
   (b) Set the variable, SENSOR_VARIABLE to the name of the Perl header file which contains the configuration parameters, e.g. std (leave off the .ph extension).
2. For *std.ph:*
   (a) set the variable LOGPROG to point to the location of the *tcpdump* binary.
   (b) Set the variable PROGPAR to indicate the interface which will be used to siphon of passing IP packets. It will be set to "-i eth0" if eth0 is the "invisible" interface.
   (c) Set the variable LOGFILE to point to the directory into which the raw *tcpdump* files will be written. These files are potentially very large, so make sure you have plenty of room. "/LOG" is the default.
3. For *std.filter*, define a *tcpdump* filter to insure that all the traffic you want to see or search is captured from your invisible interface. The *std.filter* file furnished in the SHADOW-1.7 package contains: "ip." This filter says capture all IP packets. You may want to modify this filter to eliminate some traffic you don't want to save in your raw data files. Be careful to make this a simple filter so that the bulk of the packet headers are captured; more sophisticated filters are defined and used on the analyzer machine to highlight "interesting" traffic. See 4.2.7

4.1.8 Set Crontab and System Startup Files.

The script *sensor_driver.pl* is the main controlling process of the sensor. It should be run once per hour to end the previous hour's *tcpdump* process and start the next one. That process is driven from the system *crontab*, a mechanism under Unix/Linux for starting processes at specific times. The SHADOW release package contains a file */usr/local/SHADOW-1.7/sensor/sensor_crontab* which contains:

```
1  #
2  # SHADOW Release 1.7
3  # Last Modified 3 Jul 2001
4  #
5  # Written by Bill Ralph <RalphWD@nswc.navy.mil>
6  #
```

```
 7   # Crontab for a tcpdump Sensor.
 8   #
 9   # Insure system clock is consistent.
10   # If this is not on the internet, need a standard time source.
11   #
12   17 23 * * * /usr/bin/ntpdate time-a.nist.gov
13   #
14   # The next command insures that the hardware clock of a PC is reset
15   # to the time that the previous command set the Operating System
16   # time to.  Only meaningful for Linux or Intel x86 systems perhaps.
17   #
18   18 23 * * * /sbin/hwclock --systohc
19   #
20   # The startup of the SHADOW sensor driver, once per hour.
21   #
22   0 * * * * /usr/local/SHADOW/sensor/sensor_driver.pl std > /dev/null 2>&1
```

Line 12 uses a Network Time Protocol (NTP) process (*ntpdate),* at 11:17 pm each evening to fetch the current time from a reliable time server and set the system clock to that time. It is important that the sensor and analyzer have close to the same time on their respective system clocks. In fact, it may prove to be usefull to have your site firewall synchronized to the same time. Line 18 resets the hardware clock to the value of the system clock at 11:18 pm each evening. Line 22 starts the *sensor_ driver.pl* script at the 0th minute past each hour.

After site customization, the *sensor_ init.sh* script in the distribution should be moved. The variable SHADOW_PATH in that file needs to be reset to the location into which your SHADOW package was loaded. Then, copy that file into the */etc/rc.d/init.d* subdirectory with:

1. *cd /usr/local/SHADOW/sensor*
2. *cp sensor_init.sh /etc/rc.d/init.d/sensor*
3. *chkconfig --add sensor*

This will insure that the SHADOW *tcpdump* process will restart after each system reboot.

4.1.9 Protect Your Sensor.

Strip the system of all network services except SSH. On Red Hat Linux 7.1, this involves removing all services from */etc/xinetd.conf* and either removing all the files in the */etc/xinetd.d* directory, or adding the line "disable = yes" to each. The *tcpwrappers* system is furnished with Red Hat Linux as an additional security mechanism. *Tcpwrappers* allows you to put access control lists on the various services that are allowed to connect to your system. Since the

sensor needs only the SSH service to connect, the */etc/hosts.allow* file should contain something like the following, where 172.16.77.1/255.255.255.255 is replaced with the IP address and mask of your analyzer:

```
#
# hosts.allow   This file describes the names of the hosts which are
#               allowed to use the local INET services, as decided
#               by the '/usr/sbin/tcpd' server.
#
#To permit access from specific hosts only:

          sshd:        172.16.77.1/255.255.255.255:       ALLOW
         sshdfwd-X11:   172.16.77.1/255.255.255.255:       ALLOW
          ALL: ALL: DENY
```

The sensor may be vulnerable to compromise if it resides outside your firewall. You do not want to make it easy to compromise, and you certainly don't want anything on this machine to give away secrets about the rest of your network. In the SHADOW release package, and included as Appendix C is an example *sensor_rc.iptables* file used to protect the SHADOW sensors at NSWC. Customize the first part with the names of the active and passive Ethernet interfaces, and the name or subnet of your analyzer(s). The script will allow IP communications on the SSH port between the analyzer(s) and the sensor, and only those other services that may be necessary, such as DNS. It will block and log all other attempts to connect to the sensor. It will also add a layer of invisibility to your passive Ethernet interface, in that it prohibits ANY packets from being sent out from that interface.

4.1.10 Locate Your Sensor.

At this point, the sensor is pretty much configured, at least until the analyzer is ready to connect. The sensor should be moved to the point at which you want the IP traffic to be captured. The invisible interface should be placed on a hub or the span port of a switch directly outside your firewall if you want to see who is knocking at your door, or directly inside if you want to see who got through the firewall. Traditionally, the area outside the firewall is called a DMZ (for De-Militarized Zone). You may want two sensors, one outside and one inside to measure the effectiveness of the firewall.

Note: If you connect your sensor to a regular switch port, you will see no traffic except broadcast traffic on the network segment on which your invisible interface is placed. Consult your local neighborhood networking guru for help in setting up a "span" port to allow promiscuous collection of IP packets.

The sensor is ready, the next step is to build the analyzer.

4.2 How to Build a SHADOW Analyzer.

4.2.1 Analyzer Hardware Requirements.

The SHADOW analyzer machine is intended to be the workhorse of the SHADOW system. Possibly located in a DMZ, the sensor will be capturing packets to disk. The sensor should be "dumb" so that if it is compromised, the bad guys won't automatically find information about the entire network. The sensor should be untrusted, so a compromise won't provide a clear path to the inside. The sensor is strictly limited to data collection and configured to accept SSH connections only from the analyzer. The analyzer initiates those connections, retrieves the raw data files, cleans up the sensor's disk space, and provides a remote channel for sensor maintenance. The analyzer takes the raw data, filters it to produce web pages populated with "events of interest," serves those web pages via Apache, provides tools for analysts to search for more information about those events and for building and sending incident reports. So the faster and more powerful you can build your analyzer, the better off your SHADOW installation will be.

Note: The question is frequently raised about building a single box to act as both sensor and analyzer. Of course, it's possible. But remember, you are building an intrusion detection system to enhance your network security. You don't really want your intrusion detection system to add security risks. The analyzer has a lot of software installed on it, and it may have trust relationships that put other machines inside your firewall at risk. With separate sensors and analyzers, a compromised sensor adds very little security risk to your network. With them combined, the risk is significantly greater. If you want to combine the boxes, do so at your own peril and without our help.

The CPU should be the fastest you can afford. Machines with ratinigs of 350-800 MHz have been used with great success. Of course, the 800 MHz performs significantly better. So far, NSWC has used no machines with over 128 MB of memory. Since Linux can now handle larger memories, the more memory your analyzer has, the better! And fast ample disk storage is the topmost priority for the SHADOW analyzer. At NSWC, a 420 GB RAID system built on Ultra-160 SCSI controllers serves our analyzers. It holds about two months worth of raw data for three sensors, given that each sensor collects over 2 GB per day on week days. It is possible to split up the analyzer work over multiple machines. But be sure that the data access is as fast as possible; raw data searches are extremely I/O intensive. Your analyzer may be a full featured workstation, with a sophisticated video card, sound card, and all the bells and whistles, if it is to be used as the primary analysis workstation. You may lower its workload by forcing your analysts to connect to the analyzer via browser, so a high perfomance video card in the analyzer is not necessary. This is all necessarily very

vague; it depends totally on the amount of traffic your site sees, how long you want to keep the raw data around, how active the probers are in examining your site, etc. The cardinal rule for the analzer hardware is: the faster the better!

4.2.2 Install and Patch the Analyzer Operating System.

Be sure to read section 4.1.2. The SHADOW analyzer in the Red Hat installation process terminology is a server, that may be used as a workstation as well. So the "custom" configuration choice is still the best during that installation phase.

During the partition dialog, keep the following in mind. The analyzer fetches data files from the sensors, uses them to generate the web pages, and saves them for use in searches. That means that your analyzer must hold the raw data files from all your sensors and for as long as you want to be able to search it. Keep in mind the points made in section 4.2.1, the analyzer requires fast and commodious disk resources. How you partition the disk is almost a matter of personal preference. At NSWC, the raw data files have been allocated a separate disk farm (RAID). The operating system and other files are stored on the system disk.

At the point where the OS install procedure asks about selecting packages to install, make sure to include Apache, *iptables*, XFree86, Gnome and/or KDE. Be sure to verify that Perl and the development tools are also included. You can exclude the risky stuff you shouldn't use for security reasons: rlogind, rshd, rexecd, ftpd, tftpd, linuxconf, telnetd, etc.

After the operating system and basic utilities are installed, go to Red Hat Updates `http://updates.redhat.rpm/` and download the updates for your version of the distribution. These are the patches, including security fixes, to the system and utilities that have been made since the distribution was released. ***IT IS VERY IMPORTANT TO INSTALL THESE FIXES AND TO PERIODICALLY MONITOR THE SITE FOR ADDITIONAL ONES!!!!*** Computer and network security present a constantly changing environment, if you assume that your are up to date with your security, someone out there is working diligently to prove you wrong. Once you've collected all the RPMs (Red Hat Package Manager, the files in which Red Hat furnishes its software), into a directory, use the rpm command to update your operating system:

```
rpm -Fvh *.rpm
```

Note that you should not fetch and apply updated RPMs for the kernel. The kernel will be built and customized to your hardware in step 4.2.3 below.

4.2.3 Build a Custom Analyzer Kernel.

In step 4.1.3 above, a custom kernel was built for the sensor. The objective was to configure it such that only those software components which are absolutely necessary on the sensor are included in the kernel. The operating system on the sensor is not expected to do much; keep *tcpdump* running to collect and save the raw data files. The analyzer is a different story. It fetches data and runs scripts to produce and serve web pages. It also provides tools to allow searches of the raw data and perfoms maintenance on the sensor. Nevertheless, it makes sense to optimize the kernel and eliminate those drivers for which you don't have the hardware.

Note: From this point on, this section is a duplicate of 4.1.3. If you got it there, you may not need it here.

To optimize the performance of the Linux kernel for your specific hardware, build a custom kernel. The latest version of the Linux kernel should be downloaded from Kernel.org `www.kernel.org` to insure that it has the latest fixes and capabilities.

Here are the steps to build a custom kernel:

1. *Download the latest kernel, e.g. linux-2.4.9.tar.gz.*
2. *rm /usr/src/linux*
3. *cd /usr/src; tar xvfz /some/path/linux-2.4.9.tar.gz*
4. *ln -s linux-2.4.9 linux*
5. *cd /usr/src/linux*
6. *make menuconfig (or make xconfig if using X Windows)*

At this point, a menu will appear that presents you with all the possible hardware and software configurations. Answer 'N' to all hardware not on your system. Answer 'N' to all software unnecessary for your analyzer, e.g. amateur radio, telephony, ISDN, etc. When you complete your configuration, at the end of the menu, click on Exit (Menuconfig) or "Save and Exit (Xconfig), which creates a file "*.config*" containing all the configuration options you selected during the process. See B for a copy of a *.config* file that specifies the kernel configuration for an analyzer at NSWC.

Once your configuration file matches your hardware and desired software, build the new kernel and put it into place by:

1. *cd /usr/src/linux*

2. *make dep*
3. *make bzImage*
4. *make modules*
5. *make modules_install*
6. *cp System.map /boot/System.map-{$VERSION}*
7. *cp arch/i386/boot/bzImage /boot/vmlinuz-{$VERSION}*
8. Edit the file */etc/lilo.conf* to include the new kernel you just built.
9. Run */sbin/lilo* to re-install the boot loader with the new kernel information.
10. Re-boot your system to see if the new kernel will run.

    Where $VERSION is the version of the kernel you are building, e.g. 2.4.9.

Be sure to study the document on how to build a Linux kernel, which can be found at: Kernel-HOWTO http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html.

4.2.4 Create SHADOW User Account.

On the analyzer, the Apache web server should not run as a privileged user, so a new user account should be created to be the owner of the SHADOW data files, the Apache *httpd* processes, and the web tree. More details are spelled out in the section 4.2.6. The name "shadow" seems to fit this user.

To create the "shadow" user. As root:

```
useradd -c "Lamont Cranston, the SHADOW" -u 666 -d /home/SHADOW shadow
```

By default, Red Hat Linux automatically creates a group with the same name as the user, and *bash* as the default shell. Do not give "shadow" a password. As an additional security measure, no one should be able to login directly to the analyzer as "shadow." Later, in section 4.2.6 "shadow" will be given some *OpenSSH* keys with with to communicate with the sensors and fetch their data.

4.2.5 Unpack the SHADOW-1.7 Software

The SHADOW software comes in a gzipped tarball: SHADOW-1.7.tar.gz. Choose a location for unpacking the software. "*/usr/local/*SHADOW" might be a good place, so you can unpack the tarball in that location:

1. *mkdir -p /usr/local/SHADOW*
2. *cd /usr/local/SHADOW*
3. *tar xvfz /wherever/you/stuck/the/tarball/SHADOW-1.7.tar.gz*

4.2.6 Install the Accessories.

Inside the SHADOW-1.7 directory is an accessories subdirectory which contains the pre-packaged accessories that SHADOW depends on to work. They are collected in subdirectories *rpms*, *tarballs*, *patches*, and *specs*. The *specs* directory contains *spec* files for *tcpdump* and *OpenSSH*. A *spec* file is the RPM specification file for building RPMs. The *patches* directory contains a patch to *tcpdump* as explained in section 4.2.6. Also included in the *patches* directory is a patch to a generic Apache *httpd.conf* file to assist in configuring Apache. The following subsections outline the steps necessary to install each of the accessories.

Install and Configure Apache. Apache is furnished with the Red Hat distribution. It will be installed with the rest of the operating system if you select it during the installation process. You can install the *mod_ssl* and *mod_perl* modules as well during installation to assist Apache. *Mod_ssl* adds the processing necessary for Secure Sockets Layer, and *mod_perl* adds perl processing to the Apache daemon itself, thus speeding up the execution of CGI scripts written in Perl. Apache needs some SHADOW specific configuration. In the */usr/local/SHADOW-1.7/accessories/patches* directory is a file called *httpd_conf.patch* created by *diff* that illustrates the differences between a SHADOW configured *httpd.conf* file and the one released with Red Hat Ver. 7.1. The file lists three lines of context before and after the actual differences. Given that this file is not too out of date with the version of Apache you have, you can modify your Apache configuration file with the steps:

1. `cd /usr/local/SHADOW/accessories/patches`
2. `Manually edit the httpd_conf.patch file to use your IP addresses and domain names.`
3. `cd /etc/httpd/conf`
4. `patch -p1 < /usr/local/SHADOW/accessories/patches/httpd_conf.patch`

If the patch fails, you can manually update your */etc/httpd/conf/httpd.conf* by reading it and changing the lines corresponding to the lines in the patch file. Be sure to specify your local addresses and domain names in the actual *httpd.conf* file. Restart Apache with */etc/rc.d/init.d/httpd restart*.

Install the SHADOW Home Web Page. The *httpd.conf* file installed in section 4.2.6, by default, expects the SHADOW home page to be located in */home/shadow/html*.

Create the home page by:

1. `mkdir -p /home/shadow/html/tcpdump_results`

2. `cd /usr/local/SHADOW/httpd/home`
3. `cp * /home/shadow/html`
4. `cp .htaccess /home/shadow/html`
5. `chown -R shadow:shadow /home/shadow`

Install the Compress::Zlib Perl Module. The sensor runs *gzip* to compress the raw data files as they are being collected. After the analyzer has fetched the data files it must *gunzip* them before running *tcpdump* with the filters to generate the web pages. Since the filters got so complicated at NSWC, *tcpdump* couldn't parse them all at once, so they were split into smaller files. The result was that the raw data files were unzipped repeatedly, once for each filter. With SHADOW-1.7, the *fetchem.pl* script was modified to unzip the raw data file once and feed the output to multiple *tcpdump* processes each running a different filter file. Another speed-up can be obtained by installing the *Compress::Zlib* Perl module. This module makes *gunzip* available to Perl as a subroutine call rather than a call to the operating system to spawn a separate process to run *gunzip*. However, not every SHADOW user has the experience or knowledge to install a Perl module, so the scripts were programmed to detect if the *Compress::Zlib* module is available, and use it if it is If it is not installed, the Perl scripts will use the system spawn method of running *gunzip*.

To install the *Compress::Zlib* module:

1. *cd /usr/local/SHADOW/accessories/tarballs*
2. *tar xvfz Compress-Zlib-1.11.tar.gz*
3. *cd Compress-Zlib-1.1.1*
4. Read the README file: *more README*
5. *perl Makefile.PL*
6. *make*
7. *make test*
8. If no errors have occurred, then type *make install* and you are done. If you got errors, search through the README for clues as to how to recover.

Install Tcpdump on the Analyzer. See the section 4.1.5 for information about why it is necessary to install a different version of *tcpdump* than the one furnished by Red Hat.

Install *tcpdump* with the following command:

```
cd /usr/local/SHADOW/accessories/rpms
rpm -Uvh --force tcpdump-3.6.2-wdr2.i386.rpm
```

The source, patches, and spec files are also furnished, so non-Red Hat Linux users can compile and install *tcpdump* as well.

Install and configure OpenSSH. SHADOW needs a mechanism for the analyzer to contact the sensor and fetch the data files that the sensor creates. But because the sensor may be located in a DMZ, outside the protection of the site firewall, that transport mechanism required some security. Otherwise, anyone who might be able to plant a sniffer in the DMZ would be able to pick up passwords for accounts on the sensor. And because the data being fetched by the analyzer is a snapshot of the network traffic for the whole site, it would be risky to transport that data in the clear. SSH was the mechanism chosen to answer to this problem. The SSH protocol encrypts the traffic between communicating machines using sophisticated battle-proven Open Source encryption algorithms. In addition, SSH adds a layer of authentication between client and server to protect against spoofing. For reasons alluded to in section 4.1.6, *OpenSSH* is the chosen SSH implementation for SHADOW. Included in the SHADOW package is *OpenSSH* version 2.9.9p1 in an rpm. Install it by:

cd */usr/local/SHADOW/accessories/rpms*
rpm -Uvh --force /openssh-2.9.9p1-wdr1.i386.rpm

If it balks, you may have to uninstall the version of *OpenSSH* included with Red Hat Linux:

1. *rpm -e openssh openssh-server openssh-clients openssh-askpass*
2. *rpm -e openssh-askpass-gnome*

As part of the installation, *OpenSSH* generates three host keys */etc/ssh/ssh_host_key,* */etc/ssh/ssh_host_rsa_key,* and */etc/ssh/ssh_host_dsa_key.* These are the host's private keys; the public key counterparts are */etc/ssh/ssh_host_key.pub,* */etc/ssh/ssh_host_rsa_key.pub,* and */etc/ssh/ssh_host_dsa_key.pub,* respectively.

The public host keys on the sensors need to be copied to the */etc/ssh* directory on the analyzer as follows. (There are other mechanisms for getting the sensors' public keys to the analyzer, this is one example.)

1. Copy */etc/ssh/ssh_host_key.pub* on the sensors to */etc/ssh/ssh_known_hosts* on the analyzer. This is for SSH protocol version 1 compatibility, and is largely deprecated now because of identified security problems.
2. Copy both */etc/ssh/ssh_host_rsa_key.pub* and */etc/ssh/ssh_host_dsa_key.pub* on the sensors to */etc/ssh/ssh_known_hosts2* on the analyzer. This is for SSH protocol version 2, which has been found to be more robust. The *ssh_host_rsa_key* is generated using the RSA algorithm and *ssh_host_dsa_key* was generated using the DSA algorithm. See [SSH].
3. On each of the sensors, insert a blank floppy into your floppy drive.
4. cd /etc/ssh
5. cat ssh_host_key.pub > /tmp/ssh_known_hosts
6. cat ssh_host_dsa_key.pub ssh_host_rsa_key.pub > /tmp/ssh_known_hosts2
7. cd /tmp

8. `tar cvf /dev/fd0 ssh_known_hosts ssh_known_hosts2`

On the analyzer:

1. `Insert the floppy.`
2. `cd /etc/ssh`
3. `tar xvf /dev/fd0`

Note: Here's a quick description of why this is needed. As part of the SSH protocol, an incoming connection request is received by the sensor's *sshd* daemon which then transmits a copy of the sensor's public key to the originator of the connection request, the analyzer. The analyzer compares the public key he receives from the sensor to a stored value kept in the */etc/ssh/ssh_known_hosts2* file. If they match, the analyzer is sure he is connected to the correct machine. Otherwise, someone could be attempting to impersonate the sensor or intercept communications between the sensor and analyzer. This mechanism, called "public key authentication," decreases the possibility of that happening.

In section 4.2.4 above, a user account named "shadow" was created to be the owner of the processes and files within the SHADOW and Apache systems. Shadow now needs to have a private/public key pair created so that the shadow account is allowed to connect from the analyzer to the sensors to fetch the raw data files. The analyzer host itself is known to the sensor by the contents of the *ssh_known_hosts* and *ssh_known_hosts2* files that were created above. Generation of shadow's keys is done by:

Become root on the analyzer then "*su - shadow*" to become the "shadow" user.

1. *`mkdir .ssh`*
2. *`chmod 600 .ssh`*
3. *`/usr/bin/ssh-keygen -b 1024 -t rsa1 -f .ssh/id_rsa1.`*
   `When prompted for a passphrase, enter return and repeat.`
4. *`/usr/bin/ssh-keygen -b 1024 -t dsa -f .ssh/id_dsa.`*
   `When prompted for a passphrase, enter return and repeat.`
5. *`/usr/bin/ssh-keygen -b 1024 -t rsa2 -f .ssh/id_rsa2.`*
   `When prompted for a passphrase, enter return and repeat.`
6. *`cd .ssh`*
7. *`cat id_rsa1.pub > authorized_keys`*
8. *`cat id_dsa.pub id_rsa2.pub > authorized_keys2`*
9. `Insert a floppy.`

   `tar cvf /dev/fd0 authorized_keys authorized_keys`

On the sensor:

1. Insert the floppy.
2. *cd /root*
3. *mkdir .ssh*
4. *chmod 600 .ssh*
5. *cd .ssh*
6. *tar xvf /dev/fd0*

Steps 3, 4, and 5 above generate SSH keys for user shadow. Step 3 generates an SSH protocol 1 password; step 4 generates an SSH protocol 2 password using the DSA algorithm; and step 5 generates an SSH protocol 2 password using the RSA algorithm. All three keys are not needed; in the process of migrating from protocol 1 to 2, so a protocol 1 key pair is retained for compatibility with older sensors. You will note that there is no passphrase on the shadow's private keys. As you'll see in section 4.2.7, the SHADOW scripts will be run as user shadow from an automated *crontab* script each hour. If the shadow private SSH key has a passphrase defined, then the script will halt to await entry of that passphrase to complete each connection from the analyzer to the sensor. By defining an empty passphrase, that situation is avoided, at a small additional risk. Creating the user shadow without the ability for anyone to log in directly as shadow, lessens that risk somewhat. **Your analyzer will need to be protected, because user shadow on your analyzer has permission to log in as root on your sensors without a password.**

SSH is particularly picky about permissions on its files and directories. Make sure that the *.ssh* directories, and all files contained in those directories, in /root and /home/shadow are owned by root and shadow respectively and have permissions 700 on the directories and 600 on the files. In step 4.2.6 below, the SSH connection between the analyzer and the sensors will be tested to insure that the OpenSSH software is configured and operating correctly.

Note: Here's a high level explanation of the use of "shadow's" SSH keys. On the sensor, the file */root/.ssh/ssh_ authorized_ keys2* contain copies of the public keys of any users allowed to connect as root to the sensor. Remember that private/public key encryption depends on the fact that a message encrypted with one half of a key pair can *ONLY* be decrypted by the other key in that pair. So if the sensor can decrypt a message using shadow's public key in his file, it could only have been encrypted by the corresponding private key, the one belonging to shadow on the analyzer.

Once again, further clarification of the SSH process can be obtained in the book cited in Ref. [SSH].

Test the Analyzer to Sensor Connection. Let's review. At this stage in the installation process, a two computer SHADOW system is under construction, consisting of a sensor and an analyzer. The shadow account on the analyzer will automatically run an hourly script, (see section 4.2.9), to connect to the sensor and fetch the previous hour's gzipped raw *tcpdump* data file, then run some filters on the data and generate a web page of the "interesting" events. Obviously, one of the crucial aspects of this relationship is the ability of the analyzer to connect to the sensor, fetch a file, and run some commands on the sensor. OpenSSH has been chosen for our transport mechanism. OpenSSH components have been installed on both machines. A "shadow" user account has been created on the analyzer. Some passphrase-free SSH keys for that account have been generated, with the public keys copied to the sensor. Now before the SHADOW system can become functional, the connection from the shadow account on the analyzer to the sensor must be tested and verified operational.

1. Become *root* on the analyzer; then type: *su - shadow* to become shadow.

2. Type: *ssh -v -l root sensor01.goodguys.com* , where sensor1.goodguys.com is the name of the sensor to which to connect. The -v option will produce a lot of debugging information, verbosly documenting each step through the SSH protocol. Where it finds problems, it will print messages that should help in solving them. If you get the sensor's root login prompt, congratulations! You have successfully tested and verified the ability of the analyzer to connect to the sensor. If not, proceed with the next debugging step.

3. If this attempt fails, look at the permission flags on the sensor of the */root/.ssh* and the */etc/ssh* directories. The directories themselves should be owned by root with the following permissions:

```
    ls -ld /root/.ssh /etc/ssh
drwxr-xr-x    2 root     root         1024 Mar 28 13:48 /etc/ssh
drwx------    2 root     root         1024 Mar  1 11:37 /root/.ssh
```

   The files should look like:

```
    ls -l /root/.ssh
-rw-------    1 root     root         1387 Feb 21 15:01 authorized_keys
-rw-------    1 root     root         1701 Feb 21 15:02 authorized_keys2
-rw-------    1 root     root          651 Sep 20  1999 known_hosts
-rw-------    1 root     root          512 Sep 20  1999 random_seed
ls -l /etc/ssh
-rw-------    1 root     root        26287 Mar 28 13:17 primes
-rw-r--r--    1 root     root         1042 Mar 20 09:30 ssh_config
```

```
-rw-------    1 root     root          668 Jul  7  2000 ssh_host_dsa_key
-rw-r--r--    1 root     root          615 Jul  7  2000 ssh_host_dsa_key.pub
-rw-------    1 root     root          542 Aug 13  1998 ssh_host_key
-rw-r--r--    1 root     root          346 Aug 13  1998 ssh_host_key.pub
-rw-------    1 root     root          887 Mar  6 14:48 ssh_host_rsa_key
-rw-r--r--    1 root     root          210 Mar  6 14:48 ssh_host_rsa_key.pub
-rw-r--r--    1 root     root         2168 Jul  7  2000 ssh_known_hosts
-rw-r--r--    1 root     root          814 Mar  1 12:11 ssh_known_hosts2
-rw-------    1 root     root          512 Jul  7  2000 ssh_random_seed
-rw-------    1 root     root         1695 Mar 20 09:33 sshd_config
```

After you reset the permissions on the files and directories, try step 2 again.

4. The final recommendation for debugging the analyzer to sensor connection is to go to the sensor and type */etc/rc.d/init.d/sshd stop*. Then try */usr/sbin/sshd -d*, which will start the *sshd* daemon in debug mode. By attempting the connection from the analyzer of step 2 above, the *sshd* daemon on the sensor will write debugging messages similar to those of the *ssh -v* shown in step 2, but from the perspective of the other end of the connection. This information should help debug connections that fail.

5. If all else fails, dash to your local bookstore and purchase a copy of [SSH].

Install and Configure NMAP. One of the tools available on the SHADOW toolbar allows the analyst to run *nmap* to obtain some reconnaissance about an IP address or system name of interest. If it's a system within your domain, you can ascertain what ports are open to help identify why that particular system is a target of outside probes. It is not advisable for you to use *nmap* on machines outside your purview. *Nmap* is a valuable tool, but it can be disruptive to some systems. Use it with great care.

Here is how to install the *nmap* software from the tarball:

1. `cd /usr/local/SHADOW/accessories/tarballs`
2. `tar xvfz nmap-2.53.tgz`
3. `cd nmap-2.53`
4. `./configure`
5. `make`
6. `su root`
7. `make install`

See section 4.2.7 for configuring the *nmap.cgi* script to use *nmap*.

Configure SUDO. *SUDO* (superuser do) is a command furnished with Red Hat Linux in their Powertools collection. This command allows an ordinary user to execute a particular command or set of commands with root privileges. It provides extensive logging of the account and the command it attempts to use. Since the user "shadow" is the owner of the *httpd* process, when someone clicks the "NMAP" button on their SHADOW toolbar, the *nmap.cgi* script is run as shadow. In order to utilize all the power of *nmap,* root privileges are required. *Sudo* gives shadow root privileges to execute the *nmap* command.

Here is the configuration of the /etc/sudoers file:

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for the details on how to write a sudoers file.
#

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL
shadow  saprobe=NOPASSWD: /usr/local/bin/nmap
```

The last line of the file gives shadow permission to execute "*/usr/local/bin/nmap*" without providing a password. *Sudo* provides a lot more capability that is needed for SHADOW, see the man page and [Unix Admin]. To prevent just anyone from using *nmap,* Apache must be configured to require authentication for users who click on the NMAP key, see section 4.2.7.

4.2.7 Install and Configure SHADOW software.

New to SHADOW Version 1.7 is a system wide configuration file. An example is furnished in the distribution tarball in */usr/local/SHADOW/etc/SHADOW.conf*:

```
 1  #  SHADOW master configuration file.  Used by the Perl scripts to orient
 2  #  themselves.
 3  #
 4  #  /usr/local/etc/SHADOW.conf   - SHADOW Version 1.7
 5  #                                 Last changed 21 Jun 2001
 6  #
 7  $SHADOW_PATH = "/usr/local/SHADOW";
 8  #
 9  # The paths to the SHADOW sub-commands are here.  Modify these if you want to
10  # specify a path different than "which" will find.
11  #
12  $SSH_CMD = `which ssh`;
13  $SCP_CMD = `which scp`;
```

```
14  $TCPDUMP_CMD = `which tcpdump`;
15  $GUNZIP_CMD = `which gunzip`;
16  $GZIP_CMD = `which gzip`;
17  chomp($SSH_CMD, $SCP_CMD, $TCPDUMP_CMD, $GUNZIP_CMD, $GZIP_CMD);
18  #
19  # Paths to various SHADOW components.
20  #
21  $SHADOW_SENSOR_PATH =  "$SHADOW_PATH/sensor";
22  $SHADOW_SITE_PATH = "$SHADOW_PATH/sites";
23  $SHADOW_FILTER_PATH = "$SHADOW_PATH/filters";
24  $SHADOW_CGI_PATH = "$SHADOW_PATH/httpd/cgi-bin";
25  $SHADOW_RAW_DATA_PATH = "/home/shadow";
26  $SHADOW_WEB_PAGES_PATH = "$SHADOW_RAW_DATA_PATH/html/tcpdump_results";
27  #
28  $SHADOW_REL_WEB_PAGES_ROOT = "/tcpdump_results";
29  #
30  $SHADOW_IR_SEQNO_FILE = "$SHADOW_CGI_PATH/IR_seq";
31  $SHADOW_IR_DATA_FILE = "/var/spool/SHADOW/Incident-Reports";
```

This is a Perl header file, (normally a ".ph") file that is read by every one of the SHADOW analyzer scripts to access some globally defined parameters. The file, or a link to it, is expected in the location */usr/local/etc/SHADOW.conf*. You can tailor this file to reflect where your SHADOW files are stored. Line 7 defines where you want SHADOW to look for the unpacked tarball; where the scripts are stored. Lines 12-16 define the locations of the commands *ssh, scp, tcpdump, gunzip,* and *gzip.* As written above, the script will find the commands providing they may be found in the paths specified by your PATH environment variable. If you have located these commands in non-standard places for some reason, insert the paths to the commands here. For lines 21-31, various subdirectories of the SHADOW distribution are defined, where the SHADOW tarball places them. If you move any of these subdirectories, be sure to change */usr/local/etc/SHADOW.conf* to let the SHADOW scripts find the right paths or create appropriate symbolic links.

Note: The */usr/local/etc/SHADOW.conf* file is **NOT** used by the sensor scripts, for this version of SHADOW. For your sensors, you need not worry about building this file. See section 4.1.7.

Create the Required Directories. When the SHADOW tarball was unpacked, it created the necessary subdirectories under the */usr/local/SHADOW* directory. Other directories need to be created and ownership given to shadow:

1. `mkdir -p /home/shadow/{SITES}` [or the value of $SHADOW_RAW_DATA_PATH from */usr/local/etc/SHADOW.conf*] for each of your sensors, (a sensor = a site) to hold the raw data on the analyzer.

2. `mkdir -p /home/shadow/html/tcpdump_results/{SITES}` [or the value of $SHADOW_WEB_PAGES_PATH from */usr/local/etc/SHADOW.conf*] for each of your sensors to hold the web pages.

3. `chown -R shadow:shadow` */home/shadow*

4. `mkdir -p /var/spool/SHADOW/Incident-Reports [or the value of $SHADOW_IR_DATA_FILE from /usr/local/etc/SHADOW.conf ]`

5. `chown shadow:shadow /var/spool/SHADOW/Incident-Reports`

6. `mkdir /usr/local/SHADOW/filters/{SITES} [or the value of $SHADOW_FILTER_PATH from /usr/local/etc/SHADOW.conf ]`

7. `chown -R shadow:shadow /usr/local/SHADOW`

Configure the Individual Sensor (Site) Configuration Files. In the subdirectory */usr/local/SHADOW/sites,* an entry needs to be configured for each sensor to which the analyzer is going to regularly connect. Configure the entries in this file to reflect information particular to each sensor. The "Site" that is used in the name of the file and the subdirectory that the raw data and web pages are stored is some word that you create to identify the particular sensor that is the source of the data, such as perhaps, the sensor's name. The sample Site1.ph furnished in the SHADOW distribution is included:

```
#  Variables needed by the analyzer scripts.  Tailor this file
#  to define the paths for different sensor sites.
#
# Site.ph            - SHADOW Version 1.7
#                      Last changed 9 Jul 2001
#
use POSIX qw(strftime);
use Time::Local;
#
# The '$SITE" name is a name used to identify a sensor subdirectory.  This name
# will be used to create subdirectories under the analyzer "/LOG" directory
# and the home page of the web pages that SHADOW creates to display the data.
#
->$SITE="Site1";
#
# Include a list of IP network addresses that are considered "inside" your site.
# (Not currently used in SHADOW-1.7)
#
->@SITE_IP=("172.16.31", "192.168");
#
# Put here the name of the machine on which the SHADOW sensor software is
# located.  The analyzer fetches the raw data from the sensor.
#
->$SENSOR="sensor01.goodguys.com";
#
# Put the name of the machine which runs Apache to serve up your SHADOW
# generated web pages.
#
->$WEB_SERVER="www.goodguys.com";
#
# Change the following line to reflect the directory on your sensor in which
# the raw sensor data is stored.
#
->$SENSOR_DIR="/LOG";
#
# The following line reflects the directory on your analyzer machine
```

```
# into which the raw sensor data is fetched.  The variable $SHADOW_RAW_DATA_PATH
# is defined in /usr/local/etc/SHADOW.conf.
#
$ANALYZER_DIR="$SHADOW_RAW_DATA_PATH/$SITE";
#
# The following line reflects the directory on your analyzer machine where
# SHADOW will create the web pages which hold the filtered data.  The variable
# $SHADOW_WEB_PAGES_PATH is defined in /usr/local/etc/SHADOW.conf.
#
$OUTPUT_WEB_DIR="$SHADOW_WEB_PAGES_PATH/$SITE";
#
# The following variable reflects the relative path from the DocumentRoot
# variable defined in the Apache configuration files to the actual html files.
# The variable $SHADOW_REL_WEB_PAGES_ROOT is defined in
# /usr/local/etc/SHADOW.conf.
#
$URL_OUTPUT_DIR="$SHADOW_REL_WEB_PAGES_ROOT/$SITE";
#
# The following line reflects where the filters for the site $SITE are
# stored.  The variable $SHADOW_FILTER_PATH is defined in
# /usr/local/etc/SHADOW.conf.
#
$FILTER_DIR="$SHADOW_FILTER_PATH/$SITE";
#
# The SCAN_THRESHOLD is the number of different IPs that a "foreign" machine
# can contact before SHADOW lists that foreign machine as a possible scanner.
# Change it to reflect your preference.
#
$SCAN_THRESHHOLD = "7";
#
# Set the following variable to the number of days you want to keep the
# raw data files on your sensor's disks before the cleanup.pl script removes
# them.  It depends on the sizes of your files, the amount of sensor disk space,
# and your taste.
#
$CLEAN_DATE = 2;
```

You must create one "Site.ph" file for each sensor from which your analyzer will obtain data. Not much needs to be changed in the file once the system wide */usr/local/etc/SHADOW.conf* has been created. You can tweak the parameters in these sensor specific files once the system is fully operational.

Configure the *Tcpdump* Filters. In the */usr/local/SHADOW/filters* directory are seven files: *filter.getall.doc, goodhost.filter.doc, icmp.filter.doc, ip.filter.doc, tcp.filter.doc, and udp.filter.doc*. These are the "documented" filter files. You need to edit each file and replace the generic IP addresses with the IP addresses of your particular site. There is one file for each IP protocol. The *goodhost.filter.doc* covers specific purpose machines in your site, e.g. your mail server, or your web servers. The *filter.getall.doc* is used by the *find_ scan.pl* script to identify those IP addresses considered to be "inside" your site. After you have edited each file, strip the comments out by using the *comment_ strip* script, for example:

```
/usr/local/SHADOW/comment_strip ip.filter.doc > /usr/local/SHADOW/filters/{SITE}/ip.filter
/usr/local/SHADOW/comment_strip icmp.filter.doc > /usr/local/SHADOW/filters/{SITE}/icmp.filter
/usr/local/SHADOW/comment_strip tcp.filter.doc > /usr/local/SHADOW/filters/{SITE}/tcp.filter
/usr/local/SHADOW/comment_strip udp.filter.doc > /usr/local/SHADOW/filters/{SITE}/udp.filter
/usr/local/SHADOW/comment_strip goodhost.filter.doc > /usr/local/SHADOW/filters/{SITE}/goodhost.filter
/usr/local/SHADOW/comment_strip filter.getall.doc > /usr/local/SHADOW/filters/{SITE}/filter.getall
```

A set of filters must be configured for each of the sensors (sites) to which the analyzer will connect. The filters may differ as to what kind of traffic to watch for. For example, you may want a sensor outside your firewall to watch for incoming *telnet* connections, while you may want another one inside to ignore them. After each filter is created, you can verify that the *tcpdump* command will parse it by:

```
tcpdump -i eth0 -n -F /usr/local/SHADOW/filters/{SITE}/XXX.filter
```

If that command does not give a "parse error," then the filter will run. Whether it will put those items you intended on the web page is another matter. The filter files provided are copies of the production filters in use at NSWC, with site specific information removed. Perhaps the biggest task a SHADOW analyst has to do is periodically modify the filters to reflect the traffic that is "normal" at his site in order to isolate the abnormal traffic on the web page as events of interest. Books have been written to provide assistance in configuring the *tcpdump* filters. References [NID] and [Intrusion Signatures] are good ones. Go to `www.sans.org` for training in Intrusion Detection and assistance in configuration of "signatures."

Configure the CGI Scripts. In the directory */usr/local/SHADOW/httpd/cgi-bin,* are the CGI (Common Gateway Interface) scripts used by the SHADOW system to generate web pages for the various tools offered for use. Configuration of the scripts included in the tarball is described in the paragraphs below.

Configure the compose_IR.cgi Script. The script compose_IR.cgi is used to build an Incident Report (IR) and mail copies to interested parties. Below are excerpts from the file *compose_IR.cgi*, with line numbers. All the lines preceded by "->" need to be modified to suit your site, changing the name of the company, possibly using different incident types, etc. This script uses two mail aliases, Obf_IRs and Raw_IRs. Raw_IRs is defined at NSWC as a mailing list of folks to receive the raw incident reports with the actual addresses of the local machines. Obf_IRs is a list of mail recipients that receive the same report except that the script */usr/local/SHADOW/obfuscate.pl* has been run on it. That script modifies the IP addresses and domain names within the Incident Report to conceal actual addresses while conveying information about the nature of incident. When you modify the appropriate lines, be sure to leave the surrounding punctuation intact.

```
     1   #!/usr/bin/perl
     2   #
     3   # compose_IR.cgi          - SHADOW Release 1.7
     4   #                           Last Changed 11 Jul 2001
     5   #
     6   # Written by:               Bill Ralph
     7   #                           <RalphWD@nswc.navy.mil>
     .   .   .
    32   sub fetch_seqno
    33   {
    34   # Read a file for an IR sequence number for todays date.  Do not modify the
    35   # file at this point.  That will be done when the form is submitted.
    36   #
    37   # Written by Bill Ralph - 25 Jan 1999
    38   #
    39   #
    40   # Calculate the IR sequence number from todays date.
    41   #
    42      $todays_date = strftime("%Y%m%d", localtime);
-> 43      $todays_prefix = "GGI-IDR${todays_date}";
    44      $todays_seq = "001";
    .   .   .
    54   }
    55   #
    56   sub update_seqno
    57   {
    58   # Read a file for an IR sequence number for todays date.  If it exists,
    59   # increment and resave it.  If not create the file, create the seq no, and
    60   # save it.
    61   #
    62   # Written by Bill Ralph - 12/29/98
    63   #
    64   #
    65   # Calculate the IR sequence number from todays date.
    66   #
    67      $todays_date = strftime("%Y%m%d", localtime);
-> 68      $todays_prefix = "GGI-IDR${todays_date}";
    69      $todays_seq = "001";
    .   .   .
   103   if (!param) {
   104      $shadow_seqno = fetch_seqno();
   105      print start_form(-target=>'_self'),
-> 106       h3({-align=>CENTER}, "GoodGuys Industries - Network Detection Report"),
   107      "GoodGuys Report No.:  ",
   108      textfield(-name=>'rep_num',
   109                -size=>26,
   110                -value=>$shadow_seqno),
   111      p(),
   112      "Actual Addresses Mail Recipients:  ",
   113      textfield(-name=>'raw_mailto',
   114                -size=>30,
-> 115                 -value=>'Raw_IRs'),
   116      p(),
   117      "Obfuscated Addresses Mail Recipients:  ",
   118      textfield(-name=>'obf_mailto',
   119                -size=>30,
-> 120                 -value=>'Obf_IRs'),
```

```
  121      p(),
  122      ol({-type=>"1"},
  123         li("Report Date:  $today"),
  124         p(),
  125         li("Incident Date:  "),
  126         textfield(-name=>'inci_date',
  127                   -size=>20),
  128         p(),
  129         li("Type of Incident:  "),
  130         scrolling_list(-name=>'inci_type',
  131                        -size=>1,
  132                        -values=>[
-> 133                                'Denial of Service Attempt',
-> 134                                'IMAP Connection Attempt',
-> 135                                'Remote Login Attempt',
-> 136                                'RESET Scan',
-> 137                                'SYN/RST Scan',
-> 138                                'FTP Scan',
-> 139                                'Port Scan',
-> 140                                'POP3 Scan',
-> 141                                'SNMP Probe/Scan',
-> 142                                'RPC/Portmap Connection Attempt',
-> 143                                'DNS Zone Transfer Attempt',
-> 144                                'Single System Connection Attempt',
-> 145                                'SOCKS Exploit',
-> 146                                'Informational Report',
-> 147                                'Multiple Target/Port Scan or Connection Attempts',
-> 148                                'Network Mapping Attempt',
-> 149                                'Unknown Probe type',
-> 150                                'ICMP Scan',
-> 151                                'DNS Scan',
-> 152                                'NETBIOS Scan',
-> 153                                'Unknown UDP Event',
-> 154                                'Unknown ICMP Event',
-> 155                                'Unknown TCP Event',
-> 156                                'Possible IP Spoofing Event',
  157                                ],
-> 158                         -default=>'Informational Report'),
  159         p(),
  160         li("Subjects Involved:  "),
  161         p(),
  162         ol({-type=>"a"},
  163            li("Source:  "),
  164            textfield(-name=>'inci_source', -size=>35),
  165            p(),
  166            li("Target(s):  "),
  167            textfield(-name=>'inci_target', -size=>50,
-> 168                    -value=>"GoodGuys Industries Incorporated - Podunk HQ"),
  169         ),
  170         p(),
  171         li("Location of Detector:  "),
  172         scrolling_list(-name=>'detector',
  173                        -size=>1,
  174                        -values=>[
-> 175                            'Inside Facility Firewall',
-> 176                            'Inside Facility Perimeter, Outside Firewall',
-> 177                            'Outside Facility Firewall, Outside Perimeter'
-> 178                          ],
-> 179                         -default=>'Inside Facility Firewall'),
  180         p(),
```

```
  .  .  .
  206     $inci_source = param('inci_source');
  207     $inci_target = param('inci_target');
-> 208     $mail_subject = "$rep_num  :  ${src_txt}${inci_type} \@ GoodGuys.com";
  209     $inci_cost = param('inci_cost');
  210     $inci_summ = param('summary');
  211     $inci_summ =~ tr/\r//d;
  212     $raw_mail_cmd = "/usr/sbin/sendmail -t -oi";
  213     $obf_mail_cmd = "/usr/sbin/sendmail -t -oi";
  214  #  $raw_mail_cmd = "cat > /tmp/raw_mail_$$";
  215  #  $obf_mail_cmd = "cat > /tmp/obf_mail_$$";
  216  #
  217  # Create an array of lines of our mail message.
  218  #
  219     @lines = split(/\n/, <<"EOF" );
  220  Subject:  $mail_subject
  221
-> 222                    GoodGuys Industries - Network Security Division
-> 223                              Network Detection Report
  224
-> 225                                 Phone 000-555-1212
  226
-> 227   GGII Intrusion Detection Report No.:  $rep_num
  228
  229     1.  Report Date:  $today
  230     2.  Incident Date:  $inci_date
  231     3.  Type of Incident:  $inci_type
  232     4.  Subjects Involved:
  233           a.  ${src_txt}Source:  $inci_source
  234           b.  Target(s):  $inci_target
  235     5.  Location of Detector:  $detector
  236     6.  Cost of this Incident:  $inci_cost
  237     7.  Summary of Incident and Investigation Results:
  238
  239  $inci_summ
  240
-> 241   ***** End of GGI Intrusion Detection Report No.:  $rep_num *****
  242  EOF
  .  .  .
  262  #
  263  # Insert the "To:" line for the raw and obfuscated mail arrays.
  264  #
-> 265      unshift(@lines, "From:  The GoodGuys SHADOW Team <shadow\@goodguys.com>");
-> 266      unshift(@lines, "To:  $raw_recipients");
-> 267      unshift(@obf_lines, "From:  The GoodGuys SHADOW Team <shadow\@goodguys.com>");
-> 268      unshift(@obf_lines, "To: $obf_recipients");
  269  #
  270  # Copy the array of uncensored lines to the sendmail command and the
  271  # database of Incident Reports we're saving.
  .  .  .
  302  #
  303  # print the web page to show user the mail was sent.
  304  #
  305     print
  306           a({-name=>'print', -href=>'javascript:window.print()'},
  307           img({-align=>'right',-src=>'/images/print.gif',-border=>'0'})),
-> 308            h3({-align=>CENTER},"GoodGuys Industries - Network Security Division"),
  309           h3({-align=>CENTER}," Intrusion Detection Report"),
  310           p(),
```

```
  311           hr(),
  312           center(strong("Phone 000-555-1212")),
  313           p(),
-> 314            center(strong("GGII Intrusion Detection Report No.:")," $rep_num"),
  315           p(),
  316           p(),
  .  .  .
  344   }
  345   print end_html;
```

Configure the search.cgi Script. The *search.cgi* script is the script that is run when the SEARCH button is selected from the SHADOW tool window. It displays the form that the analyst fills out to search the raw data files; executes the search and displays the results back in the search window. Below are excerpts from the file *search.cgi*, with line numbers. All the lines preceded by "->" may need to be modified to suit your site. These changes only affect the strings identifying your sensors so that an analyst can select the proper sensor data to view on the browser web page, e.g. data from "Sensor01" will be accessed from the list "Outside Site Perimeter", etc. You need to adjust the number of sensors, and how the pull down menu will identify each on the search web page and tools window.

```
    1   #!/usr/bin/perl
    2   #
    3   # search.cgi              - SHADOW Release 1.7
    4   #                           Last Changed 31 May 2001
    5   #
    6   use CGI qw/:standard/;
    7   use CGI::Carp qw(fatalsToBrowser);
    8   use POSIX qw(strftime);
    9   use Time::Local;
   10   use IO::Handle;
   11   #
   12   do "/usr/local/etc/SHADOW.conf" ||
   13      die("Unable to load SHADOW configuration file:  /usr/local/etc/SHADOW.conf.");
.  .  .
   48   %param_info = (
   49                  site=> {
   50                          param_label => "Which sensor:  ",
   51                          max_field_size => "8",
   52                          param_type => "list",
   53                          values =>["Sensor1", "Sensor2", "Sensor3"],
   54                          labels => {
-> 55                            "Sensor1" => "Outside Site Perimeter",
-> 56                            "Sensor2" => "Inside Perimeter, Outside Site Firewall",
-> 57                            "Sensor3" => "Inside Site Firewall"  ,
   58                                  },
-> 59                          default_value => "Sensor1"
   60                         },


. . .
  490   print end_html;
```

Configure the nmap.cgi Script. Sections 4.2.6, and 4.2.6 alluded to using *nmap* from the SHADOW toolbar. *Sudo* was installed to give the web browser account (shadow) permission to use the root *nmap* features. But it is desirable to restrict *nmap* usage even further; not everyone who can access and examine the SHADOW data should be allowed to run *nmap* as root. That restriction can be accomplished by placing a file named *.htaccess* in the directory where the *nmap.cgi* script is located, (*/usr/local/SHADOW/httpd/cgi-bin/privileged* by default).

Here are the contents of the .htaccess file provided in the SHADOW tarball:

```
AuthType Basic
AuthName "Privileged SHADOW Users"
AuthUserFile /usr/local/SHADOW/httpd/cgi-bin/privileged/nmap_pwd
Satisfy any
require valid-user
order deny,allow
allow from 172.16.47
deny from all
```

This file accomplishes the following: It allows anyone from the "privileged" subnet, 172.16.47, to execute any command in this directory, i.e. *nmap.cgi.* For those not on the privileged subnet, if the user has a valid name/password combination in the file *nmap_ pwd* in the same directory, he can run the *nmap.cgi* script as well. All other users who attempt to execute the script will be denied. To create user/password pairs in the file *nmap_ pwd,* you use the *htpasswd* command that is furnished as part of the Apache distribution.

```
htpasswd -c /usr/local/SHADOW/httpd/cgi-bin/privileged/nmap_pwd ImaUser
```

This command will create the *nmap_ pwd* file and add the user "ImaUser" to it, prompting for a password twice. Use the command without the "-c" option to add additional users. Please see reference [Unix Admin] for more details about how to set up Apache authentication for individual subdirectories.

Configure the Statistics Scripts. New with version 1.7 of the SHADOW Package is a set of scripts that is run to generate a daily page of IP statistics that are collected from the raw SHADOW data files. A button on the SHADOW tool window allows the SHADOW analyst to access the previous day's statistics summary. In upcoming section 4.2.9, the last entry in the analyzer crontab file is the command that generates the daily statistics summary web page. In order to generate the daily statistics, the file */usr/local/SHADOW/statistics.ph* will need to be configured to recognize inside and outside IP addresses. The lines that need changing are preceded by "->" in the listing below.

Here is the appropriate portions of the file:

```
    1   #!  /usr/bin/perl
    2   #
    3   # statistics.ph             - SHADOW Version 1.7
    4   #                             Last changed 23 May 2001
    5   #
    6   #
    7   #  Script to read a raw tcpdump hourly file, look at the packets,
    8   #  and produce some statistics about the traffic seen in that file.
    9   #  Optionally uses Compress::Zlib Perl module to directly read gzipped files.
   10   #
   11   #  Written by Bill Ralph <RalphWD@nswc.navy.mil>
. . .
  173   #
  174   # Define an array of "internal" IP addresses.
  175   #
  176   @internal_ip = (
-> 177                   "172.21.0.0", "172.22.0.0", "172.16.22.0",
  178                 );
  179   @internal_mask = (
-> 180                    "255.255.0.0", "255.255.0.0", "255.255.255.0",
  181                   );
  182   #
.  .  .
  192   #
  193   # Let's look at some specific TCP and UDP ports.
  194   #
-> 195    @TCP_ports = ( "12345", "12346", "31337", "2049", "2766", "5232", "6000",
-> 196                   "6667", "20432");
-> 197    @UDP_ports = ( "1080", "2049", "3128", "6970", "7070", "18753", "20433",
-> 198                   "31337");
  557   1;
  558   #  End of statistics.ph
```

Note: The statistics file is only created after all the twenty-four hourly files have been created, once per day. So when selecting the particular statistics file to view, remember that the statistics for today has not yet been created. Yesterday's file is the most recent one available for viewing.

4.2.8 Test the SHADOW Scripts.

At this point, your SHADOW system should be set up. To verify it, you need to manually test the SHADOW scripts to witness that everything works as intended. Hopefully, some time back, you started one of your sensors and it has been dutifully collecting hourly gzipped *tcpdump* files. Test your SHADOW analyzer by running:

```
/usr/local/SHADOW/fetchem.pl -l sensor01 -d YYYYMMDD -debug
```

where YYYY=year, MM=2-digit month, and DD=2-digit day of the month.

This command will run the *fetchem.pl* script which will:

1. Create a debug file */tmp/fetchem.log*
2. Load the site (sensor) specific information from the header file, */usr/local/SHADOW/sites/sensor01.ph.*
3. Create the subdirectories on the analyzer to hold the raw data file and the output html file, if necessary.
4. Use SSH to see if the necessary raw file exists on the sensor and abort if it doesn't.
5. Use *scp* to copy the gzipped raw data file from the sensor to its place on the analyzer.
6. Create an output HTML file and write the necessary header information to it.
7. Start gunzipping the raw data file, and feed the gunzipped data to as many *tcpdump* processes as necessary for each of the filters in the sensor filter directory, plus one more for the *find_scan.pl* script.
8. Collect all the output text files from the execution of the *tcpdump* processes into a single text file.
9. Sort the concatenated text file by IP address, and convert the IP addresses to system names where possible.
10. Copy the sorted and resolved text file to the HTML file.
11. Add the information from the *find_scan.pl* script to the HTML file, then add the navigation bar to the end along with the necessary closing HTML statements.
12. Remove all the temporary text files created by this run and exit.

With the "-debug" flag set on the *fetchem.pl* script call, logging information will be written to /tmp/fetchem.log at many places in the script. By examining each line in that log file, you can find the corresponding place in the *fetchem.pl* script where the message was written. If the script aborts for some reason, the lines in the log will allow you to find the area of the script that failed. When the process completes normally, you will have an HTML file created in:

```
/home/shadow/tcpdump_results/{SITE}/Mondd/YYYYMMDDHH.html
```

for example: */home/shadow/tcpdump_results/sensor01/July 27/2001072713.html.*

4.2.9 Configure the Crontab and System Startup Files.

The SHADOW release package contains a file */usr/local/SHADOW-1.7/analyzer_ crontab.shadow* which contains:

```
1  #             SHADOW Version: 1.7
2  #             Last Changed:   12 Jul 2001
3  #
4  # Crontab for a SHADOW Analyzer.
5  #
6  # The following entries should not be done as root.
7  #
8  $SHADOW_PATH=/usr/local/SHADOW
9  5 * * * * $SHADOW_PATH/fetchem.pl -l SITE
10 #
11 17 1 * * * $SHADOW_PATH/cleanup.pl -l SITE
12 #
13 19 0 * * * $SHADOW_PATH/run_daily_stats.pl -l SITE
```

This file is an example crontab that should be run be run as the user "shadow" on the analyzer. Line 9 says that at 5 minutes past each hour, the script "`/usr/local/SHADOW/fetchem.pl -l SITE`" will be run. You need a line like this with the name you have chosen for your sensor substituted for "SITE." If you have more that one sensor, you will need multiple lines, one for each, with the starting time separated by some number of minutes of your choosing. Line 11 says that the "`/usr/local/SHADOW/cleanup.pl -l SITE`" script will be run at 1:17 (am) each day. Line 13 says that the "`/usr/local/SHADOW/run_daily_stats.pl -l SITE`" script will be run at 0:19 (12:19am) each day. As in the first case, your sensor name should be substituted for SITE and multiple lines duplicated and staggered in time if you serve more than one sensor.

The SHADOW release package contains a file */usr/local/SHADOW-1.7/analyzer_ crontab.root* which contains:

```
1  #             SHADOW Version: 1.7
2  #             Last Changed:   12 Jul 2001
3  #
4  # Crontab for a SHADOW Analyzer. We net to insure that the system clock
5  # is consistent between the analyzer and the sensor.
6  # If this system is not on the internet, we need to reference a standard
7  # time source.
8  #
9  # These two entries must be done as root.
10 #
11 17 23 * * * /usr/bin/ntpdate time-a.nist.gov
12 18 23 * * * /sbin/hwclock –systohc
13 #
```

Line 11 uses the *ntpdate* program to synchronize time with the *time-a.nist.gov* time server at 23:17 (11:17pm). Line 12 says that at 23:18 the hardware clock will be set to the system clock just synchronized to the NTP time source. As stated in section 4.1.8, the system clocks of the analyzer and its sensor should be kept relatively synchronized, without being too picky about it.

4.2.10 Protect Your Analyzer.

The analyzer does not require as much protection as the sensor. Nevertheless, as part of a "security in depth" philosophy, you should set up your analyzer to detect unwanted activity and protect itself, rather than depending on outside resources. Prudent security precautions are never inappropriate. Like the sensor, strip the system of all network services except SSH. On Red Hat Linux 7.1, this involves removing all services from */etc/xinetd.conf* and either removing all the files in the */etc/xinetd.d* directory, or adding the line "disable = yes" to each. The *tcpwrappers* system furnished with Red Hat Linux as an additional security mechanism allows you to put access control lists on the various services that are allowed to connect to your system. For more help, see the *hosts.allow* and *hosts.deny* man pages and section 4.1.9.

In the SHADOW release package, and included as Appendix D is an example *sensor_rc.iptables* file used to protect the NSWC SHADOW analyzers. The script will allow IP communications on only those other services that may be necessary, such as DNS, httpd, https, SMTP, NNTP, etc. It will block and log all other attempts to connect to the analyzer.

Recommended resources for more general information about Linux security, are
[Max Linux Security, Linux Sys Security, Real World Linux Security, Linux Firewalls], and [Linux Firewalls].

4.3 Put Your SHADOW System into Production.

At this point, everything should be working. The sensor should be capturing packets and writing the raw files. The analyzer should be fetching those files, running them through your filters and generating your web pages. Your tools should be usable. The easy part of SHADOW configuration is complete. Now comes the difficult part: learning what your network traffic really looks like and adjusting your filters to reflect the reality of your situation. As you get more and more familiar with your network, your filters will undoubtedly need to be changed to reflect what you consider "risky" and what you don't. The art of creating filters for different traffic patterns is beyond the scope of SHADOW installation. The best books on the topic are [NID] and [Intrusion Signatures]. They will give you a foundation on which to build your SHADOW system, and your intrusion detection expertise. Stay in touch with SANS Institute `www.sans.org` to follow what's happening in intrusion detection and to get detailed training.

Happy detecting.

A SHADOW Sensor Kernel Configuration

This is a copy of the kernel configuration file for a SHADOW sensor (taken from a Ver. 2.4.9 kernel build). Only those configuration options selected are printed out; the rest are left out to shorten the length of the file.

```
#
# Automatically generated make config:  don't edit
#
CONFIG_X86=y
CONFIG_ISA=y
CONFIG_UID16=y

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y

#
# Loadable module support
#

#
# Processor type and features
#
CONFIG_M486=y
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_CMPXCHG=y
CONFIG_X86_XADD=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_X86_L1_CACHE_SHIFT=4
CONFIG_X86_USE_STRING_486=y
CONFIG_X86_ALIGNMENT_16=y
CONFIG_NOHIGHMEM=y
CONFIG_MTRR=y

#
# General setup
#
CONFIG_NET=y
CONFIG_PCI=y
CONFIG_PCI_GOANY=y
CONFIG_PCI_BIOS=y
CONFIG_PCI_DIRECT=y
CONFIG_PCI_NAMES=y
CONFIG_SYSVIPC=y
CONFIG_SYSCTL=y
CONFIG_KCORE_ELF=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y

#
# Memory Technology Devices (MTD)
```

```
#

#
# Parallel port support
#

#
# Plug and Play configuration
#

#
# Block devices
#
CONFIG_BLK_DEV_FD=y

#
# Multi-device support (RAID and LVM)
#

#
# Networking options
#
CONFIG_PACKET=y
CONFIG_NETFILTER=y
CONFIG_NETFILTER_DEBUG=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_SYN_COOKIES=y

#
#   IP: Netfilter Configuration
#
CONFIG_IP_NF_CONNTRACK=y
CONFIG_IP_NF_FTP=y
CONFIG_IP_NF_IPTABLES=y
CONFIG_IP_NF_MATCH_LIMIT=y
CONFIG_IP_NF_MATCH_MAC=y
CONFIG_IP_NF_MATCH_MARK=y
CONFIG_IP_NF_MATCH_MULTIPORT=y
CONFIG_IP_NF_MATCH_TOS=y
CONFIG_IP_NF_MATCH_STATE=y
CONFIG_IP_NF_FILTER=y
CONFIG_IP_NF_TARGET_REJECT=y
CONFIG_IP_NF_TARGET_LOG=y

#
#
#

#
# QoS and/or fair queueing
#

#
# Telephony Support
#

#
# ATA/IDE/MFM/RLL support
#
CONFIG_IDE=y
```

```
#
# IDE, ATA and ATAPI Block devices
#
CONFIG_BLK_DEV_IDE=y

#
# Please see Documentation/ide.txt for help/info on IDE drives
#
CONFIG_BLK_DEV_IDEDISK=y
CONFIG_BLK_DEV_IDECD=y

#
# IDE chipset support/bugfixes
#
CONFIG_BLK_DEV_IDEPCI=y
CONFIG_IDEPCI_SHARE_IRQ=y
CONFIG_BLK_DEV_IDEDMA_PCI=y
CONFIG_BLK_DEV_ADMA=y
CONFIG_IDEDMA_PCI_AUTO=y
CONFIG_BLK_DEV_IDEDMA=y
CONFIG_IDEDMA_AUTO=y

#
# SCSI support
#
CONFIG_SCSI=y

#
# SCSI support type (disk, tape, CD-ROM)
#
CONFIG_BLK_DEV_SD=y
CONFIG_SD_EXTRA_DEVS=40

#
# Some SCSI devices (e.g.  CD jukebox) support multiple LUNs
#
CONFIG_SCSI_CONSTANTS=y

#
# SCSI low-level drivers
#
CONFIG_SCSI_AIC7XXX=y
CONFIG_AIC7XXX_CMDS_PER_DEVICE=8
CONFIG_AIC7XXX_RESET_DELAY_MS=5000

#
# Fusion MPT device support
#

#
# IEEE 1394 (FireWire) support (EXPERIMENTAL)
#

#
# I2O device support
#

#
# Network device support
#
```

```
CONFIG_NETDEVICES=y

#
# ARCnet devices
#

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y
CONFIG_NET_VENDOR_3COM=y
CONFIG_VORTEX=y
CONFIG_NET_PCI=y
CONFIG_TULIP=y

#
# Ethernet (1000 Mbit)
#

#
# Wireless LAN (non-hamradio)
#

#
# Token Ring devices
#

#
# Wan interfaces
#

#
# Amateur Radio support
#

#
# IrDA (infrared) support
#

#
# ISDN subsystem
#

#
# Old CD-ROM drivers (not SCSI, not IDE)
#

#
# Input core support
#

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_SERIAL=y
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256

#
```

```
# I2C support
#

#
# Mice
#
CONFIG_MOUSE=y
CONFIG_PSMOUSE=y

#
# Joysticks
#

#
# Game port support
#

#
# Gameport joysticks
#

#
# Serial port support
#

#
# Serial port joysticks
#

#
# Parallel port joysticks
#

#
#   Parport support is needed for parallel port joysticks
#

#
# Watchdog Cards
#
CONFIG_RTC=y

#
# Ftape, the floppy tape device driver
#

#
# Multimedia devices
#

#
# File systems
#
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_ISO9660_FS=y
CONFIG_JOLIET=y
CONFIG_PROC_FS=y
CONFIG_DEVPTS_FS=y
CONFIG_EXT2_FS=y
```

```
#
# Network File Systems
#

#
# Partition Types
#
CONFIG_MSDOS_PARTITION=y
CONFIG_NLS=y

#
# Native Language Support
#
CONFIG_NLS_DEFAULT="cp437"
CONFIG_NLS_CODEPAGE_437=y
CONFIG_NLS_ISO8859_1=y

#
# Console drivers
#
CONFIG_VGA_CONSOLE=y

#
# Frame-buffer support
#

#
# Sound
#

#
# USB support
#

#
# Bluetooth support
#

#
# Kernel hacking
#
```

B SHADOW Analyzer Kernel Configuration

This is a copy of the kernel configuration file for a SHADOW analyzer (taken from a Ver. 2.4.9 kernel build). Only those configuration options selected are printed out; the rest are left out to shorten the length of the file.

```
#
# Automatically generated make config:  don't edit
#
CONFIG_X86=y
CONFIG_ISA=y
CONFIG_UID16=y

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y

#
# Loadable module support
#
CONFIG_MODULES=y
CONFIG_KMOD=y

#
# Processor type and features
#
CONFIG_M686=y
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_CMPXCHG=y
CONFIG_X86_XADD=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_X86_L1_CACHE_SHIFT=5
CONFIG_X86_TSC=y
CONFIG_X86_GOOD_APIC=y
CONFIG_X86_PGE=y
CONFIG_X86_USE_PPRO_CHECKSUM=y
CONFIG_NOHIGHMEM=y
CONFIG_MTRR=y

#
# General setup
#
CONFIG_NET=y
CONFIG_PCI=y
CONFIG_PCI_GOANY=y
CONFIG_PCI_BIOS=y
CONFIG_PCI_DIRECT=y
CONFIG_PCI_NAMES=y
CONFIG_SYSVIPC=y
CONFIG_BSD_PROCESS_ACCT=y
CONFIG_SYSCTL=y
CONFIG_KCORE_ELF=y
CONFIG_BINFMT_AOUT=y
```

```
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y

#
# Memory Technology Devices (MTD)
#

#
# Parallel port support
#

#
# Plug and Play configuration
#

#
# Block devices
#
CONFIG_BLK_DEV_FD=y
CONFIG_BLK_DEV_LOOP=m

#
# Multi-device support (RAID and LVM)
#

#
# Networking options
#
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_NETFILTER=y
CONFIG_NETFILTER_DEBUG=y
CONFIG_FILTER=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_SYN_COOKIES=y

#
#   IP: Netfilter Configuration
#
CONFIG_IP_NF_CONNTRACK=y
CONFIG_IP_NF_FTP=y
CONFIG_IP_NF_IPTABLES=y
CONFIG_IP_NF_MATCH_LIMIT=y
CONFIG_IP_NF_MATCH_MAC=y
CONFIG_IP_NF_MATCH_MARK=y
CONFIG_IP_NF_MATCH_MULTIPORT=y
CONFIG_IP_NF_MATCH_TOS=y
CONFIG_IP_NF_MATCH_STATE=y
CONFIG_IP_NF_FILTER=y
CONFIG_IP_NF_TARGET_REJECT=y
CONFIG_IP_NF_TARGET_LOG=y

#
#
#

#
# QoS and/or fair queueing
#
```

```
#
# Telephony Support
#

#
# ATA/IDE/MFM/RLL support
#
CONFIG_IDE=y

#
# IDE, ATA and ATAPI Block devices
#
CONFIG_BLK_DEV_IDE=y

#
# Please see Documentation/ide.txt for help/info on IDE drives
#
CONFIG_BLK_DEV_IDEDISK=y
CONFIG_BLK_DEV_IDECD=y
CONFIG_BLK_DEV_IDESCSI=m

#
# IDE chipset support/bugfixes
#
CONFIG_BLK_DEV_IDEPCI=y
CONFIG_IDEPCI_SHARE_IRQ=y
CONFIG_BLK_DEV_IDEDMA_PCI=y
CONFIG_BLK_DEV_ADMA=y
CONFIG_IDEDMA_PCI_AUTO=y
CONFIG_BLK_DEV_IDEDMA=y
CONFIG_IDEDMA_AUTO=y

#
# SCSI support
#
CONFIG_SCSI=y

#
# SCSI support type (disk, tape, CD-ROM)
#
CONFIG_BLK_DEV_SD=y
CONFIG_SD_EXTRA_DEVS=40
CONFIG_CHR_DEV_ST=m
CONFIG_BLK_DEV_SR=m
CONFIG_BLK_DEV_SR_VENDOR=y
CONFIG_SR_EXTRA_DEVS=2
CONFIG_CHR_DEV_SG=m

#
# Some SCSI devices (e.g.  CD jukebox) support multiple LUNs
#
CONFIG_SCSI_DEBUG_QUEUES=y
CONFIG_SCSI_MULTI_LUN=y
CONFIG_SCSI_CONSTANTS=y
CONFIG_SCSI_LOGGING=y

#
# SCSI low-level drivers
#
CONFIG_SCSI_AIC7XXX=y
CONFIG_AIC7XXX_CMDS_PER_DEVICE=8
```

```
CONFIG_AIC7XXX_RESET_DELAY_MS=5000

#
# Fusion MPT device support
#

#
# IEEE 1394 (FireWire) support (EXPERIMENTAL)
#

#
# I2O device support
#

#
# Network device support
#
CONFIG_NETDEVICES=y

#
# ARCnet devices
#

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y
CONFIG_NET_VENDOR_3COM=y
CONFIG_VORTEX=y

#
# Ethernet (1000 Mbit)
#

#
# Wireless LAN (non-hamradio)
#

#
# Token Ring devices
#

#
# Wan interfaces
#

#
# Amateur Radio support
#

#
# IrDA (infrared) support
#

#
# ISDN subsystem
#

#
# Old CD-ROM drivers (not SCSI, not IDE)
#
```

```
#
# Input core support
#

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_SERIAL=y
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256

#
# I2C support
#

#
# Mice
#
CONFIG_MOUSE=y
CONFIG_PSMOUSE=y

#
# Joysticks
#

#
# Game port support
#

#
# Gameport joysticks
#

#
# Serial port support
#

#
# Serial port joysticks
#

#
# Parallel port joysticks
#

#
#   Parport support is needed for parallel port joysticks
#

#
# Watchdog Cards
#
CONFIG_RTC=y

#
# Ftape, the floppy tape device driver
#
CONFIG_AGP=m
```

```
CONFIG_AGP_INTEL=y
CONFIG_DRM=y
CONFIG_DRM_R128=m

#
# Multimedia devices
#

#
# File systems
#
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_TMPFS=y
CONFIG_ISO9660_FS=y
CONFIG_JOLIET=y
CONFIG_MINIX_FS=m
CONFIG_NTFS_FS=m
CONFIG_PROC_FS=y
CONFIG_DEVPTS_FS=y
CONFIG_EXT2_FS=y

#
# Network File Systems
#
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
CONFIG_NFSD=y
CONFIG_NFSD_V3=y
CONFIG_SUNRPC=y
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_SMB_FS=y

#
# Partition Types
#
CONFIG_MSDOS_PARTITION=y
CONFIG_SMB_NLS=y
CONFIG_NLS=y

#
# Native Language Support
#
CONFIG_NLS_DEFAULT="cp437"
CONFIG_NLS_CODEPAGE_437=y
CONFIG_NLS_ISO8859_1=y

#
# Console drivers
#
CONFIG_VGA_CONSOLE=y

#
# Frame-buffer support
#

#
# Sound
#
```

```
CONFIG_SOUND=m
CONFIG_SOUND_ES1371=m

#
# USB support
#

#
# Bluetooth support
#

#
# Kernel hacking
#
```

C Protecting your SHADOW sensor

The following is an example *iptables* shell script that can be put into the */etc/rc.d/rc.local* script on the sensor to help protect it from unauthorized access attempts.

```
#!/bin/sh
#
# SHADOW Version 1.7
# Last Modified 3 Jul 2001
#
# Written by Bill Ralph <RalphWD@nswc.navy.mil>
#
# Script to limit packets accepted by a Linux box.
#
LOOPBACK="127.0.0.1"
LOOPBACK_IF="lo"
ACTIVE_IF="eth0"
PASSIVE_IF="eth1"
ANALYZER="172.16.47.1/32"
#
# Source function library.
.  /etc/rc.d/init.d/functions

if [ !  -f /etc/sysconfig/network ]; then
    exit 0
fi

.  /etc/sysconfig/network

# Check that networking is up.

[ ${NETWORKING} = "no" ] && exit 0

[ -x /sbin/ifconfig ] || exit 0

CWD=‘pwd‘
cd /etc/sysconfig/network-scripts

[ -f /etc/sysconfig/network-scripts/ifcfg-$ACTIVE_IF ] || exit 0

.  /etc/sysconfig/network-scripts/ifcfg-$ACTIVE_IF
#

# ----------------------------------------------------------------------------
#
# Define some other variables.
#
LOG_LEVEL="notice"
#
#
# ----------------------------------------------------------------------------
#
# First, flush any current chains, then remove any user-defined chains.
#
 iptables -F
 iptables -X
#
# ----------------------------------------------------------------------------
```

```
#
# Make the default policy of the input chain DROP
#
 iptables -P INPUT DROP
#
# ------------------------------------------------------------------------
#
# Allow unlimited traffic on loopback interface.
#
 iptables -A INPUT  -i lo -j ACCEPT
 iptables -A OUTPUT -o lo  -j ACCEPT
 iptables -A INPUT  -i $ACTIVE_IF -s $IPADDR/32 -j ACCEPT
 iptables -A OUTPUT -o $ACTIVE_IF -s $IPADDR/32 -j ACCEPT
#
#-----------------------------------------------------------------
#
# Allow nothing to go out on our passive interface.
#
 iptables -A OUTPUT -i $PASSIVE_IF -j DROP
#
# ------------------------------------------------------------------------
#
# Create a chain to log information about a connection, then drop it.
#
 iptables -N log_drop
 iptables -A log_drop -m limit --limit 5/m \
                 -j LOG --log-level $LOG_LEVEL --log-prefix "Dropped:  "
 iptables -A log_drop -j DROP
#
# ------------------------------------------------------------------------
#
# Create a chain to reject a packet with a icmp-host-prohibited packet.
# Severerly limit the bandwidth accepted.
#
 iptables -N log_rej
 iptables -A log_rej -m limit --limit 1/h \
                 -j LOG --log-level $LOG_LEVEL --log-prefix "Rejected:  "
 iptables -A log_rej -j REJECT --reject-with icmp-host-prohibited
#
# ------------------------------------------------------------------------
#
# Create a chain to specifically accept IP traffic on established
# connections that we initiated.  Log traffic on INVALID connections.
#
 iptables -N allowed
 iptables -A allowed --match state --state INVALID -j LOG \
                  --log-prefix "INVALID conn:"
 iptables -A allowed --match state --state INVALID -j DROP
 iptables -A allowed --match state --state ESTABLISHED,RELATED -j ACCEPT
#
# ------------------------------------------------------------------------
#
# Create a chain to look at TCP packets.  Examine the flags and limit the
# weird ones we will accept.  Limit the average matching rate to slow down
# nmaps and such.
#
 iptables -N tcp_pkts
 iptables -A tcp_pkts -p tcp --tcp-flags ALL FIN,URG,PSH -m limit \
                --limit 5/minute -j LOG --log-level $LOG_LEVEL \
                --log-prefix "NMAP-XMAS:"
 iptables -A tcp_pkts -p tcp --tcp-flags ALL FIN,URG,PSH -j DROP
```

```
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,RST SYN,RST -m limit \
               --limit 5/minute -j LOG --log-level $LOG_LEVEL \
               --log-prefix "SYN/RST:"
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit \
               --limit 5/minute -j LOG --log-level $LOG_LEVEL \
               --log-prefix "SYN/FIN:"
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
#
# Accept SSH connections only from analyzer (subnet) - Deny all other
# SSH attempts.
#
 iptables -A tcp_pkts -p tcp -s $ANALYZER --dport 22 -j ACCEPT
#
# Accept SMTP connections from our mail relays.
#
 iptables -A tcp_pkts -p tcp -s 172.16.99.0/24 --dport 25 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.254.41/32 --dport 25 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.254.42/32 --dport 25 -j ACCEPT
#
# Log all TCP SYN (connection) requests
#
 iptables -A tcp_pkts -p tcp -i !  lo --syn -j LOG --log-level $LOG_LEVEL \
                  --log-prefix "Incoming SYN:"
 iptables -A tcp_pkts -j DROP
#
# ----------------------------------------------------------------------------
#
# Create a chain for ICMP packets.  Accept them only from our network
# except for broadcasts.
#
 iptables -N icmp_pkts
 iptables -A icmp_pkts -p icmp -d ${NETWORK}/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -d ${BROADCAST}/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -d 255.255.255.255/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -s 172.16.99.0/24 --icmp-type 8 -j ACCEPT
 iptables -A icmp_pkts -p icmp -s 0.0.0.0/0 --icmp-type echo-reply -j ACCEPT
 iptables -A icmp_pkts -j log_drop
#
# ----------------------------------------------------------------------------
#
# Create a chain for UDP packets.  Deny most, except NETBIOS from our subnet,
# and DNS from our name servers.  Don't need 'em, and they're dangerous.
# Deny RIP packets from our router.  Log all other udp packets.
#
 iptables -N udp_pkts
 iptables -A udp_pkts -p udp -s 172.16.99.0/24 --dport 111 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.99.0/24 --dport 113 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.86.1/32 --sport 123 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.100.1/32 --sport 53 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.254.2/32 --sport 53 -j ACCEPT
 iptables -A udp_pkts -p udp --sport 520 -j DROP
 iptables -A udp_pkts -j log_drop

#
#
# ----------------------------------------------------------------------------
#
# Apply the defined chains to the input chain.
#
 iptables -A INPUT -j allowed
```

```
iptables -A INPUT -p tcp -j tcp_pkts
iptables -A INPUT -p udp -j udp_pkts
iptables -A INPUT -p icmp -j icmp_pkts
```

D Protecting your SHADOW Analyzer

The following is an example *iptables* shell script that can be put into the */etc/rc.d/rc.local* script on the analyzer to help protect it from unauthorized access attempts.

```
#!/bin/sh
#
# SHADOW Version 1.7
# Last Modified 1 Aug 2001
#
# Written by Bill Ralph <RalphWD@nswc.navy.mil>
#
# Script to limit packets accepted by a Linux box.
#
# Source function library.
.  /etc/rc.d/init.d/functions

if [ !  -f /etc/sysconfig/network ]; then
    exit 0
fi

.  /etc/sysconfig/network

# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 0

[ -x /sbin/ifconfig ] || exit 0

CWD=‘pwd‘
cd /etc/sysconfig/network-scripts

[ -f /etc/sysconfig/network-scripts/ifcfg-eth0 ] || exit 0

.  /etc/sysconfig/network-scripts/ifcfg-eth0

#
# ------------------------------------------------------------------------
#
# Define some other variables.
#
LOG_LEVEL="notice"
#
#
# ------------------------------------------------------------------------
#
# First, flush any current chains, then remove any user-defined chains.
#
 iptables -F
 iptables -X
#
# ------------------------------------------------------------------------
#
# Make the default policy of the input chain DROP
#
 iptables -P INPUT DROP
#
# ------------------------------------------------------------------------
#
```

```
# Allow unlimited traffic on loopback interface.
#
 iptables -A INPUT  -i lo -j ACCEPT
 iptables -A OUTPUT -o lo  -j ACCEPT
 iptables -A INPUT  -i eth0 -s $IPADDR/32 -j ACCEPT
 iptables -A OUTPUT -o eth0 -s $IPADDR/32 -j ACCEPT
#
# ------------------------------------------------------------------------
#
# Create a chain to log information about a connection, then drop it.
#
 iptables -N log_drop
 iptables -A log_drop -m limit --limit 5/m \
                  -j LOG --log-level $LOG_LEVEL --log-prefix "Dropped:  "
 iptables -A log_drop -j DROP
#
# ------------------------------------------------------------------------
#
# Create a chain to reject a packet with a icmp-host-prohibited packet.
# Severerly limit the bandwidth accepted.
#
 iptables -N log_rej
 iptables -A log_rej -m limit --limit 1/h \
                  -j LOG --log-level $LOG_LEVEL --log-prefix "Rejected:  "
 iptables -A log_rej -j REJECT --reject-with icmp-host-prohibited
#
# ------------------------------------------------------------------------
#
# Create a chain to specifically accept IP traffic on established
# connections that we initiated.  Log traffic on INVALID connections.
#
 iptables -N allowed
 iptables -A allowed --match state --state INVALID -j LOG \
                   --log-prefix "INVALID conn:"
 iptables -A allowed --match state --state INVALID -j DROP
 iptables -A allowed --match state --state ESTABLISHED,RELATED -j ACCEPT
#
# ------------------------------------------------------------------------
#
# Create a chain to look at TCP packets.  Examine the flags and limit the
# weird ones we will accept.  Limit the average matching rate to slow down
# nmaps and such.
#
 iptables -N tcp_pkts
 iptables -A tcp_pkts -p tcp --tcp-flags ALL FIN,URG,PSH -m limit \
                --limit 5/minute -j LOG --log-level $LOG_LEVEL \
                --log-prefix "NMAP-XMAS:"
 iptables -A tcp_pkts -p tcp --tcp-flags ALL FIN,URG,PSH -j DROP
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,RST SYN,RST -m limit \
                --limit 5/minute -j LOG --log-level $LOG_LEVEL \
                --log-prefix "SYN/RST:"
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,FIN SYN,FIN -m limit \
                --limit 5/minute -j LOG --log-level $LOG_LEVEL \
                --log-prefix "SYN/FIN:"
 iptables -A tcp_pkts -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
#
# Accept SSH connections only from acceptable people - Deny all other
# SSH attempts.
#
 iptables -A tcp_pkts -p tcp -s 0.0.0.0/0 --dport 22 -j ACCEPT
```

```
#
#
# Accept TCP connections only from our subnet - Accept http/https and timed
# connections from anyone at NSWC. Deny all other
# TCP packets on the well-known ports.
#
#iptables -A tcp_pkts -p tcp -s 172.16.77.0/24 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.0.0/16 --dport 80 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.0.0/16 --dport 443 -j ACCEPT
#
# Accept SMTP connections from our friends
#
 iptables -A tcp_pkts -p tcp -s 172.16.77.0/24 --dport 25 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.100.201/32 --dport 25 -j ACCEPT
 iptables -A tcp_pkts -p tcp -s 172.16.100.202/32 --dport 25 -j ACCEPT
#
# Log all TCP SYN (connection) requests
#
 iptables -A tcp_pkts -p tcp -i !  lo --syn -j LOG --log-level $LOG_LEVEL \
                   --log-prefix "Incoming SYN:"
 iptables -A tcp_pkts -j DROP
#
# ------------------------------------------------------------------------
#
# Create a chain for ICMP packets.  Accept them only from our network
# except for broadcasts.
#
 iptables -N icmp_pkts
 iptables -A icmp_pkts -p icmp -d ${NETWORK}/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -d ${BROADCAST}/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -d 255.255.255.255/32 --icmp-type 8 -j log_drop
 iptables -A icmp_pkts -p icmp -s 172.16.77.0/24 --icmp-type 8 -j ACCEPT
 iptables -A icmp_pkts -p icmp -s 0.0.0.0/0 --icmp-type echo-reply -j ACCEPT
 iptables -A icmp_pkts -j log_drop
#
# ------------------------------------------------------------------------
#
# Create a chain for UDP packets.  Deny most, except NETBIOS from our subnet,
# and DNS from our name servers.  Don't need 'em, and they're dangerous.
# Deny RIP packets from our router.  Log all other udp packets.
#
 iptables -N udp_pkts
 iptables -A udp_pkts -p udp -s 172.16.77.0/24 --dport 111 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.77.0/24 --dport 113 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.76.1/32 --sport 123 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.77.0/24 --dport 2049 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.77.0/24 --dport 1025 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.77.0/24 --dport 135:139 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.1.2/32 --sport 53 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.1.1/32 --sport 53 -j ACCEPT
 iptables -A udp_pkts -p udp -s 172.16.77.50/32 --sport 35 -j ACCEPT
 iptables -A udp_pkts -p udp --sport 520 -j DROP
 iptables -A udp_pkts -j log_drop
#
#
# ------------------------------------------------------------------------
#
# Apply the defined chains to the input chain.
#
 iptables -A INPUT -j allowed
 iptables -A INPUT -p tcp -j tcp_pkts
```

```
iptables -A INPUT -p udp -j udp_pkts
iptables -A INPUT -p icmp -j icmp_pkts
```

References

[RHL-Install] *The Official Red Hat Linux 7.1 x86 Installation Guide,* Red Hat, Inc., 2001

[SSH] *SSH The Secure Shell, The Definitive Guide,* Daniel J. Barrett and Richard E. Slverman, ISBN: 0-596-00011-1, O'Reilly and Associates, 2001

[Unix Admin] *Unix System Administration Handbook, Third Edition,* Evi Nemeth, Garth Snyder, Scott Seebass, and Trent R. Hein, ISBN 0-13-020601-6, Prentice Hall PTR, 2001.

[Apache] *Apache, The Definitive Guide, Second Edition,* Ben Laurie & Peter Laurie, ISBN 1-56592-528-9, O'Reilly and Associates, 1999.

[Programming Perl] *Programming Perl, Third Edition,* Larry Wall, Tom Christiansen, and Jon Orwant, ISBN 0-596-00027-8, O'Reilly and Associates, 2000.

[CGI.pm] *Official Guide to Programming with CGI.pm,* Lincoln Stein, ISBN: 0-471-24744-8, John Wiley & Sons, Inc., 1998.

[Max Linux Security] *Maximum Linux Security,* Anonymous, ISBN 0-672-31670-6, SAMS Publishing, 2000.

[Linux Sys Security] *Linux System Security,* Scott Mann and Ellen L. Mitchell, ISBN 0-13-015807-0, Prentice Hall PTR, 2000.

[Real World Linux Security] *Real World Linux Security,* Bob Toxen, ISBN 0-13-028187-5, Prentice Hall PTR, 2001.

[Linux Firewalls] *Linux Firewalls,* Robert L. Ziegler, ISBN 0-7357-0900-9, New Riders Publishing, 2000.

[NID] *Network Intrusion Detection, An Analyst's Handbook, Second Edition,* Stephen Northcutt and Judy Novak, ISBN_0-7357-1008-2, New Riders Publishing, 2001.

[Intrusion Signatures] *Intrusion Signatures and Analysis,* Stephen Northcutt, Mark Cooper, Matt Fearnow, and Karen Frederick, ISBN 0-7357-1063-5, New Riders Publishing, 2001.

Index