SECOND Edition

INTRUSION DETECTION:

A Primer for Intrusion Detection Analysts

STEP BY STEP GUIDE

Stephen Northcutt and the Intrusion Detection Team
at the Naval Surface Warfare Center, Dahlgren

SANS INSTITUTE

CONTENTS

SECTION 1 Introduction And Background

Acknowledgments

We give special thanks to the folks who helped us to build and improve the software, in particular:

- The original developers of SHADOW: Stephen Northcutt, Ballistic Missile Defense Organization, Vicki Irwin, Cisco Systems, and Bill Ralph, Naval Surface Warfare Center Dahlgren Division. Stephen and Vicki are former employees of the Naval Surface Warfare Center Dahlgren Division and members of the original SHADOW development team. Bill writes the SHADOW software.
- Scott Hoye, University of Virginia, who assisted in the analysis section of this guide and was a primary tester of the system,
- Dean Goodwell, Joint National Test Facility, who improved the installation instructions and wrote an appendix on a specific installation,
- Olav Kolbu, USIT, University of Oslo, who ported some of our shell scripts to Perl,
- Pedro Vazquez, Unicamp Brazil, who has helped us throughout the project with his knowledge of tcpdump filters and their application to intrusion detection.
- Judy Novak, Army Research Lab (Jacob & Sundstrom, Inc.), who provided new filters to detect some interesting intrusion detection patterns
- Jim Matthews, Naval Surface Warfare Center Dahlgren Division, who wrote the updates for this version of the guide.

We also thank all the reviewers of this document.

Ever-growing Internet connectivity has spawned a new breed of vandals and criminals. For a variety of different motives, Internet crackers connect to machines around the world in their quest to break into sites. Stopping these attackers is challenging. Detecting when they have broken into a site or when they are attempting to break in is even more challenging.

Designed by the Naval Surface Warfare Center Dahlgren Division, SHADOW is one of the first freely available toolkits for detecting intruders. SHADOW, which is actually an acronym for Secondary Heuristics for Defensive On-line Warfare, is being used and evaluated by organizations around the United States and throughout the world for intrusion detection and network monitoring. It runs on several UNIX operating systems and can be assembled using freely available software and either existing or low-cost hardware that can be purchased for less than $10,000.

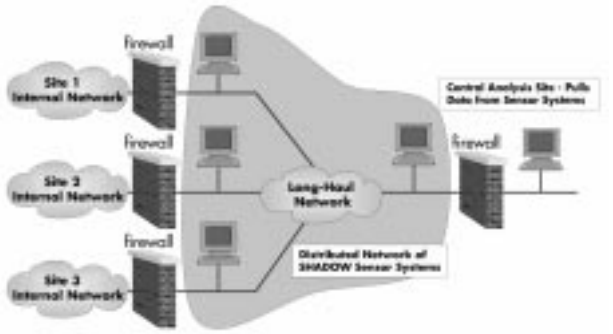
SHADOW users are generally familiar with the TCP/IP protocol and gain the greatest value from the software when they understand how to perform intrusion detection and deeper analysis to identify new types of attacks that might not be detected by other tools.

The Intrusion Detection Team at the Naval Surface Warfare Center Dahlgren Division prepared this document with the help of the SANS Institute, The Army Research Lab, and system and network administrators from all over the world. This text has is aimed toward SHADOW users and any other security professionals who feel more comfortable when they master their tool s underlying technology. The document s name, *Intrusion Detection: SHADOW Style*, connotes more than a simple manual for implementing one type of intrusion detection system. Anyone searching for a primer on intrusion detection and analysis will gain value from the Sections 4, 5, and 6, whether or not SHADOW is ever deployed at their sites. The booklet includes:

- ¥ A description of the SHADOW architecture and design considerations,
- ¥ Step by Step instructions for building your own SHADOW intrusion detection system,
- ¥ Examples of attacks and what they mean, and
- ¥ Sample filters that enable the SHADOW intrusion detects

The system needs improvement in many ways, and we are interested in your suggestions and software contributions so that SHADOW s detection technology can be more sensitive, faster, and both easier to implement and easier to use. Please send your comments and suggestions to <shadow@sans.org> with the subject SHADOW Suggestions.

SECTION 2 The Shadow Architecture



A SHADOW system includes two main parts: sensors and analysis systems. These parts could run on the same machine, but most sites separate them as shown in Figure 1. The sensors glean headers from packets that traverse a network; analysis systems pull the data from the sensor systems and study the headers looking for intrusions (detects).

SHADOW System Requirements

The computer systems used for both the sensor and the analysis systems may be any UNIX system that can compile `libpcap` and `tcpdump`. The analysis system must allow root access to the network administrator and support a recent version of `perl`, `gzip`, the `Apache` web server and Secure Shell (`ssh`) on the analysis system. The sensor system must also allow root access to the network administrator and must support `gzip` and `ssh` in addition to libpcap and tcpdump. Any of the open source UNIX-like systems (FreeBSD, NetBSD, Linux) should suffice.

Intrusion detection requires copious disk space multiple gigabytes on the sensor system and dozens or hundreds of gigabytes on the analysis system. More than half of the system cost may be tied up in mass storage.

For logistics reasons, the same hardware configuration is recommended for both the sensor and the analysis systems with the exception that the analysis system has more disk space. The reference system is a 400MHz Pentium PC, 128MB of memory, 10/100baseT network interface card, and a SCSI interface card. For sensor systems, we use a single 9GB SCSI disk drive. For the analysis system, we use two 23GB SCSI disk drives - almost 50 GB. Any large disk drive that is compatible with your UNIX systems should work. Several reviewers pointed out that organizations could save money by using non-SCSI drives. Inexpensive Pentium-class processors coupled with IDE disks enable extremely low cost systems to be built. Also, the sensor system doesn t even need a permanent monitor or video card.

SHADOW uses the tcpdump program developed by the Department of Energy as a foundation and adds support software developed by the Naval Surface Warfare Center Dahlgren Division.

SHADOW s data collection capability is based on the libpcap program developed by the Network Research Group at Lawrence Berkeley Laboratory. `Libpcap` provides an interface for application programs such as Network Flight Recorder (`NFR`) and SHADOW to read data collected by the network interface on UNIX systems. Specifically, `libpcap` is used by UNIX systems to get network information from the kernel.

Though it is possible to instruct `libpcap` to gather more (or less) information from each packet, it is common for `libpcap` to collect the first 96 bytes of a packet for analysis. The SHADOW programs examine the headers of the collected packet instead of its content. This approach to intrusion detection is called traffic, or header, analysis.

SHADOW Sensor Systems

SHADOW sensors are usually located outside an organization s firewall between the firewall and the Internet connection, an area often called the demilitarized zone (or DMZ). One of the design features of the SHADOW sensor is that it simply collects headers from all packets and has absolutely no knowledge itself of what it is looking for. This way, if the sensor were ever compromised (which is not unlikely; any system outside of your firewall should be considered a high risk), the attacker would be denied any knowledge about what information your organization is using for filtering. We recommend that you install a small network hub to support the sensor. Then, if your incident response team ever needs to plug in an additional sensor to conduct forensics analysis, you can do so quite easily.

SHADOW Analysis Systems

As shown in Figure 1, a central site has a SHADOW analysis system located inside of the site firewall. This analysis system pulls the header data collected at each sensor system on a periodic basis, usually every hour. SHADOW uses filters on the analysis system to collect events of interest such as probable attacks or probes. When queried, the analysis system displays the information to a private web page.

The web page sports a Graphical User Interface (GUI) that enables the analyst to examine information from other sensors and other time periods, to query the analysis system for more information about a host or a pattern, and to create a report suitable for sending to a Computer Incident Response Team (CIRT).

The information displayed on the web page results from tcpdump filters matching various conditions and patterns in the data. *Intrusion Detection: SHADOW Style* details a large number of example filters that can flag interesting and potentially malicious events in traffic flow.

Some patterns cannot be detected using the tcpdump filter language. Worse, some attacks that are low and slow (such as a scan that uses one packet per day) probably will not be detected with the one-hour interval normally used for packet correlation studies. An advanced, perl-based analysis tool has been included with the SHADOW toolkit to handle these kinds of problems.

SHADOW Design Considerations

SHADOW s primary requirements are (1) to detect as many attacks as possible, as easily and efficiently as possible, (2) to ease reporting the attacks to CIRTs, and (3) to support analysis of the attacks to tune defenses. We have found that all three of those objectives can be met if we pull the detects using a web browser whenever we have time and examine the kinds of probes and attacks directed against the organization. The web display is simple to use and easy to explain to new users. One person can monitor multiple sites; in fact, we monitor over ten sites.

To Push or To Pull?

Most intrusion detection systems, whether network- or host-based, push alerts when they detect an event. Their most common push methods are sending e-mail to the system administrator, sending a message to a pager, or beeping to a monitor screen. We have such capabilities in commercial systems at the Naval Surface Warfare Center Dahlgren Division, and we use them. However, we have found that in day-to-day use, these systems have serious drawbacks. Many of the alerts turn out to be false alarms. Many others turn out to be of little consequence; they are simple exploit scripts that we have already effectively blocked by our firewalls and other countermeasures. The beeping or alerting becomes so annoying that we ultimately ignore the alerts. At that point, we are relying on firewalls, system protections and other countermeasures and the intrusion detection system loses most of its value.

Traffic Analysis vs. Content Analysis for Intrusion Detection

The strength of a traffic analysis approach is that it enables us to examine and record every packet transmitted on the network. In certain situations (such as educational or health care institutions) where privacy is a critical concern, traffic analysis may be the only acceptable approach to network monitoring.

The weakness of traffic analysis, on the other hand, is that it misses certain attacks that can be detected only by examining the contents of packets to search for certain strings that indicate an attack. Analysis that uses the content of the packet rather than the header is called content analysis or sometimes string analysis.

The strength of content analysis is that it enables collection of all data from a connection once an attack string is detected. Its weakness is that, in practice, traffic volume overwhelms the systems trying to read the content. In the face of this problem, organizations that use content analysis often filter out many services, or ports, making the system and the organization blind to attacks using those particular ports or services.

About the DMZ

Throughout the history of electronic communications it has been common practice to identify the point of demarcation, the point that is the end of responsibility for one group and the beginning for another. Early firewall implementers on the Internet referred to the point of demarcation between the Internet Service Provider and an organization connected to the Internet as the Demilitarized Zone (DMZ). DMZs are fruitful locations for sensors. As a rule of thumb, however, any point of demarcation is likely to be a good location to place a sensor and careful organizations deploy multiple sensors. The SHADOW architecture easily supports multiple sensors.

A Note On Forensics

Forensics can be defined as any evidence that may be used in legal proceedings. In an intrusion situation where you are collecting evidence and have a reasonable expectation that the evidence will be used in court, it makes great sense to dedicate a sensor to evidence collection. It is entirely possible that all sensor logs and possibly even the sensor itself might be considered part of the evidence. Also, if the systems from which evidence is required are known in advance, you might wish to focus the dedicated sensor on those target system(s). This enables your organization to turn over only relevant information about all the other network traffic outside parties. Finally, a dedicated sensor might also collect the contents of the packets for more complete analysis.

Architecture Summary

In summary:

- ¥ SHADOW sensors record all traffic to and from a protected network.
- ¥ SHADOW sensors use tcpdump to record traffic.
- ¥ Each record represents a packet observed by the sensor.
- ¥ Records are bundled hourly and downloaded to the analysis system where files may be filtered for attack patterns.
- ¥ Results are displayed via a web server.

SECTION 3 Building The Shadow Systems

Step-By-Step Instructions

Step 1: Prerequisites

IMPORTANT NOTE:
Appendix A contains installation instructions specific to Red Hat Linux v5.1 for the sensor and analysis system. If you are using 5.1 or need greater detail than the information provided in the steps below, you may wish to consult Appendix A.

Action 1.1 Legal Review - Get Permission!

Deploying a sensor should only be done with approval from your organization s legal staff. All the computer systems will need warning banners and all users should be notified before the system is placed in operation. A Standard Operating Procedure detailing appropriate and inappropriate uses of the data is recommended. Determine how your site determines what constitutes acceptable use . Review or develop a firewall policy.

Action 1.2 Modifying UNIX files

While you don t really need to know UNIX in order to set up a SHADOW system, you will need to know how to use a UNIX text file editor to modify some of the SHADOW scripts. Some UNIX systems come with a graphical interface that includes an editor. Alternatively, the vi editor is available on just about every UNIX platform - learn how to use the vi editor or some equivalent editor before proceeding.

Action 1.3 Preparing for Installation

Before beginning the installation, read your UNIX vendor s documentation for specific installation tips. If you are installing on a PC system, boot your system under Windows and record the IRQ settings for all hardware peripherals. Windows allows IRQ sharing, but Linux does not. One possible solution is to physically move the network or SCSI card to another slot. You may also be able to reconfigure the jumper settings on your cards to avoid shared IRQs.

Step 2: Acquiring the SHADOW Software

The core software used to build SHADOW systems is available at no cost from the Network Research Group at Lawrence Berkeley Research Laboratory, from the Naval Surface Warfare Center Dahlgren Division, and from the Apache Group. Lawrence Berkeley Research Laboratory provides the tcpdump, libpcap, and tcpslice applications. The Naval Surface Warfare Center Dahlgren Division provides the SHADOW scripts. The Apache Group provides the Apache web server.

Secure Shell was originally developed at Helsinki University of Technology and is now distributed by SSH Communications Security Ltd., in Finland. Secure Shell is available at no cost for non-commercial sites - be sure to read the Secure Shell licensing agreements at

http://www.ssh.fi/sshprotocols2/licensing/ssh2_non-commercial_licensing.html

to determine if your site qualifies to use the free version or if the commercial version must be purchased.

Action 2.1 Obtain SHADOW tar files.

First, create a directory to hold the downloaded compressed software tar sets.

 # mkdir /usr/local/archives

 # cd /usr/local/archives

Then download the SHADOW Intrusion Detection System software from:

http://www.nswc.navy.mil/ISSEC/CID/step.tar.gz.

When the download process is complete, the /usr/local/archives directory should contain the step.tar.gz file. The SHADOW distribution includes `tcpdump`, `libpcap`, and `tcpslice` which we will extract in Step 3. These files can also be retrieved directly from

ftp://ftp.ee.lbl.gov.

Both the sensor and the analysis system will require `tcpdump` and `libpcap`, but `tcpslice` is required only on the analysis system.

Action 2.2 Acquire Secure Shell.

The SHADOW scripts are designed to use Secure Shell version 1.2.26. If you are not familiar with Secure Shell, visit the home pages at

http://ns.uoregon.edu/pgpssh/sshstart.html

and at

http://www.ssh.fi/protocols2/

to get more information. A list of Secure Shell download sites can be found at

http://www.ssh.fi/sshprotocols2/download.html.

Place the downloaded Secure Shell file, ssh-1.2.26.tar.gz, in the /usr/local/archives directory. When the download process is complete, the /usr/local/archives directory should now contain the step.tar.gz and ssh-1.2.26.tar.gz files. The sensor system runs the Secure Shell daemon and accepts ssh login requests from the analysis system. We want the sensor to trust communications from the analyzer, but not vice versa.

Action 2.3 Download Apache Web Server.

To build a SHADOW analysis system, you ll need to download and build Apache Web Server. If you re building a sensor system, you will not need to install a web server. Apache version 1.3.3 is provided in the SHADOW distribution, but we recommend that you download and install the latest version from

http://www.apache.org/dist.

Apache version 1.3.5 corrected potential buffer overflow vulnerabilities and Apache version 1.3.6 is now available. Place the downloaded Apache web server file, apache_1.3.6.tar.gz, in the /usr/local/archives directory. When the download process is complete, the /usr/local/archives directory should now contain the step.tar.gz, ssh-1.2.26.tar.gz and apache_1.3.6.tar.gz files.

Action 3.1 Obtain a computer running UNIX.

Tcpdump compiles on multiple UNIX systems. If the network you are protecting doesn t have a great deal of traffic, older obsolete workstations may be ideal. One group responsible for protecting multiple sites was able to collect a large number of Sun SPARC II pizza boxes . They fit nicely in the equipment racks and were able to handle up to full T1 speeds quite well. However, if your site has a high traffic network, you should consider using FreeBSD or BSDI UNIX. FreeBSD is considered the fastest, or most efficient, free platform for a SHADOW sensor due to FreeBSD s implementation of BPF (Berkeley Packet Filter). The sensor system should have at least 32MB of RAM and 9 GB of disk space. The machine should be dedicated to running the intrusion detection software.

Action 3.2 Install and Secure the UNIX Operating System.

Install the UNIX operating system, making sure to include a suitable C compiler. Since the primary purpose of the system is to collect data, the /LOG partition should be as large as possible and the root / partition should be about 2GB. Disable all network services except Secure Shell. Make sure that all available security patches for your system are installed. If you are unsure about what security patches are needed, a good reference site to check is

http://www.iss.net/xforce/.

When you are finished securing your system, the sensor should only be reachable by the analysis system and perhaps a backup trusted system! Modify the sensor system s /etc/hosts file to include only relevant hostname and IP address entries of your analysis system and the NIST time server:

129.6.16.35 time-a.nist.gov

198.x.x.x analysishost.analysissite.org

The sensor operating system should have the minimal possible information about the internal network. In addition to /etc/hosts, also check /etc/hosts.equiv, /etc/networks, /.rhosts yellow pages and any other files that might have configuration information about your network. Also, do not add any new accounts to this system since the analysis system will log into the root account of sensor system using Secure Shell.

Action 3.3 Unpack the SHADOW tar file.

Create the /usr/local/logger directory:

 # mkdir /usr/local/logger

Change to the /usr/local/archives directory, since this is where we will perform all of our software installation

 # cd /usr/local/archives

Unpack the SHADOW tar set:

 # tar xvfz step.tar.gz

NOTE:
If you're building an analysis system instead of a sensor system, replace the previous copy command with the following:
cp -R cid-1.5/*
/usr/local/logger

This places the SHADOW distribution into the /usr/local/archives/cid-1.5 directory and contains an INSTALL file, various README files, various SHADOW Perl scripts, and five subdirectories named accessories, filters, httpd, sensor, and sites. The accessories directory contains a few additional compressed tar files including libpcap-0.4.tar.Z, tcpdump-3.4.tar.Z, and tcpslice.tar.Z. Copy the SHADOW sensor files to the usr/local/logger/sensor directory:

 # cp -R cid-1.5/sensor* /usr/local/logger

This creates the /usr/local/logger/sensor directory which contains scripts for starting and stopping the tcpdump process.

Action 3.4 Build libpcap.

The software installation will take place from the /usr/local/archives directory:

 # cd /usr/local/archives

Unpack the libpcap tar set:

 # tar xvfZ cid-1.5/accessories/libpcap-0.4.tar.Z

Change to the new /usr/local/archives/libpcap-0.4 directory. Review the Makefile.in file and make any changes if necessary. Then build libpcap:

 # cd /usr/local/archives/libpcap-0.4
 # ./configure
 # make
 # make install
 # mkdir /usr/local/include /usr/local/include/net
 # make install-man
 # make install-incl

Action 3.5 Build tcpdump.

The software installation will take place from the /usr/local/archives directory:

 # cd /usr/local/archives

Unpack the tcpdump tar set:

 # tar xvfZ cid-1.5/accessories/ tcpdump-3.4.tar.Z

Change to the new /usr/local/archives/tcpdump-3.4 directory. Review the Makefile.in file and make any changes if necessary. Then build tcpdump:

 # cd /usr/local/archives/tcpdump-3.4
 # ./configure
 # make
 # make install
 # make install-man

Action 3.6 Build Secure Shell.

The software installation will take place from the /usr/local/archives directory:

 # cd /usr/local/archives

Unpack the Secure Shell tar set:

 # tar xvfz ssh-1.2.26.tar.gz

Change to the new /usr/local/archives/ssh-1.2.26 directory. Review the Makefile.in file and make any changes if necessary. Then build Secure Shell:

 # cd /usr/local/archives/ssh-1.2.26

 # ./configure

 # make

 # make install

Start the Secure Shell daemon:

 # /usr/local/sbin/sshd

Add the Secure Shell startup command to your UNIX system startup files, typically in /etc/rc.d/rc.local or /etc/rc.d/init.d

Action 3.7 Modify variables.ph

The /usr/local/logger/sensor/variables.ph file may need to be modified depending on your operating system. Locate the tcpdump command:

 # whereis tcpdump

which displays tcpdump: /usr/local/sbin/tcpdump on a Red Hat Linux v5.2 system. Therefore, edit the variables.ph file and modify $LOGPROC to read as follows: $LOGPROC = /usr/local/sbin/tcpdump ;

Action 3.8 Review the activities of tcpdump.driver.pl.

This is the program, run every hour by cron, that sets up some variables, calls one Perl script (stop_logger.pl) to stop the current tcpdump process and calls another Perl script (start_logger.pl) to restart tcpdump again. When stop_logger.pl executes, the previous hour s tcpdump records are written to a file named with the date and hour of its creation. After the tcpdump file is written, start_logger.pl restarts tcpdump and creates a new hourly file.

Action 3.9 Test the tcpdump.driver.pl script.

Run the script from the command line

 # /usr/local/logger/sensor/tcpdump.driver.pl > /dev/null 2>&1

Check the process table (ps ax on linux, ps -ax on SUNOS, ps -ef on ATT variants of UNIX). Check to see if tcpdump is running. Go to the /LOG directory and check to see whether the log file is present and growing. If it is, you are probably in good shape. Another file in the distribution, sensor_init.sh can be moved to /etc/rc.d/init.d under Linux to automatically restart the tcpdump process at system boot. This file is an example of the file necessary under the System V initialization method for automatically starting a process. BSD based systems will require equivalent modification of the rc.local initialization script.

Action 3.10 Configure cron.

The UNIX scheduling system, named cron, automatically runs programs, often shell or perl scripts, at a certain time. A file called sensor_crontab is included in the /usr/local/logger/sensor directory. Its contents:

17 23 * * * /usr/local/bin/rdate -s time-a.nist.gov

18 23 * * * /sbin/hwclock --systohc

0 * * * * /usr/local/logger/sensor/tcpdump.driver.pl > /dev/null 2>&1

The first entry instructs cron to run the rdate program at 5:23pm every day. The rdate program uses tcp to get the current time from a system called time-a.nist.gov. The -s switch sets the local system time to the time retrieved from the time-a.nist.gov system. The second entry of the crontab is Linux specific. It sets the hardware clock of the machine to the value of the Linux system time, which was previously synchronized with a reliable time source. Non-Linux systems may not require this entry. The final entry tells cron to execute a Perl script named tcpdump.driver.pl at the zero minute of every hour.

NOTE:
On Red Hat Linux v5.2, the "crontab -e" command is used to edit the crontab file. However, with the latest release of Red Hat Linux v5.2, the location of the vi editor changed from /usr/bin/vi to /bin/vi. In order for the "crontab -e" command to work, you must assign a symbolic link to /bin/vi as follows: **"ln -s /bin/vi /usr/bin/vi".**

Step 4. Configuring the Analysis System

Action 4.1 Obtain a UNIX computer and Secure the Operating System.

Follow the procedures in Step 3, Actions 3.1 and 3.2 to set up the operating system for your analysis system. Logically position the analyzer inside the site firewall with the sensor located outside the firewall. Add a non-privileged user account on the analysis system. The name of the account is irrelevant, but for clarity, we ll call the non-privileged account sans-analyst . Also, create a non-privileged account called httpd to be the owner of the Apache Web server processes. The home directory for the httpd account should be /home/httpd. Modify the /etc/hosts file to include hostname and IP address entries for your sensor systems and for the NIST time server:

129.6.16.35 time-a.nist.gov
198.x.x.1 sensorhost1.site1.org
198.x.x.2 sensorhost2.site2.org

Action 4.2 Build libpcap, tcpdump, and Secure Shell.

Follow the procedures in Step 3, Actions 3.3 through 3.6 to set up libpcap, tcpdump and Secure Shell on your analysis system. Note: Action 3.3 lists a different copy command (**# cp -R cid-1.5/* /usr/local/logger**) to use when building Analysis systems. This will place the SHADOW Perl scripts in the /usr/local/logger directory as well as five subdirectories named accessories, filters, httpd, sensor, and sites

Action 4.3 Configure and Test Secure Shell

Configure Secure Shell to allow password-less communication between the analyzer system and the sensor. The scripts will run automatically from a cron job every hour under the non-privileged user id created in action 4.1. The scripts must be able to run unattended and cannot halt to await interactive input of a password on the analyzer system. We want the sensor to trust communications from the analyzer, but not vice versa. There are several ways to configure Secure Shell, but we ll describe a simple method that uses an authorized_keys file on the sensor system to house public keys from the analysis system.

On the analysis system, log into the non-privileged account. Change directory to .ssh and generate public and private keys:

$ cd ~/.ssh

$ /usr/local/bin/ssh-keygen

When prompted to enter a file in which to save the key, accept the default (~/.ssh/identity).

When prompted to enter a passphrase, just press enter (i.e., do not enter a passphrase).

Then in the ~./ssh directory, a private key (identity) and public key (identity.pub) will be created.

Now transfer the public key, identity.pub to the sensor system. One method is to copy the identity.pub file to a floppy disk:

 # mount -t msdos /dev/fd0 /mnt/floppy

 # cp /home/sans-analyst/.ssh/identity.pub /mnt/floppy

 # umount /mnt/floppy

Transport the disk to the sensor system and copy the file to the root s secure shell directory. Since this is the only public key that the sensor will have, we can rename it authorized_keys. If we wanted to allow more systems to ssh login into the sensor system, then we would need to add their public keys to the authorized_keys file. On the sensor system:

 # mount -t msdos /dev/fd0 /mnt/floppy

 # cp /mnt/floppy/identity.pub /root/.ssh/identity.pub

 # cd /root/.ssh

 # mv identity.pub authorized_keys

 # umount /mnt/floppy

Note:
The Secure Shell daemon does not fork processes while it's in debug mode. This means that when the analysis system terminates a Secure Shell session, the sshd daemon on the sensor will exit if it is in debug mode.

Assuming that a network exists between the analysis system and the sensor system and that the sensor system is running the Secure Shell daemon (sshd), attempt to login from the non-privileged account on the analysis system to the root account on the sensor system:

$ ssh -l root sensor-hostname

If you run into problems, turn on verbose mode:

$ ssh -l root sensor-hostname -v

If you are still having problems, try running the Secure Shell daemon, on the sensor system, in debug mode. Kill the currently executing sshd process and then run:

 # /usr/local/sbin/sshd -debug

Action 4.4 Build tcpslice.

The software installation will take place from the /usr/local/archives directory:

 # cd /usr/local/archives

Unpack the tcpslice tar set:

 # tar xvfZ cid-1.5/accessories/ tcpslice.tar.Z

Change to the new /usr/local/archives/tcpslice-1.1a3 directory. Review the Makefile.in file and make any changes if necessary. Then build tcpslice:

 # cd /usr/local/archives/tcpslice-1.1a3
 # ./configure
 # make
 # make install
 # make install-man

Action 4.5 Install and Configure Apache Web Server

The SHADOW distribution includes Apache version 1.3.3, but Apache version 1.3.6 is now available as of this writing. Apache versions 1.3.4 and later now use a single file, httpd.conf, to configure the web server. The old configuration files, srm.conf and access.conf, are no longer required. The software installation will take place from the /usr/local/archives directory:

 # cd /usr/local/archives

Unpack the Apache tar set:

 # tar xvfz apache_1.3.6.tar.gz

Change to the new /usr/local/archives/apache_1.3.6 directory and build Apache web server:

 # cd /usr/local/archives/apache_1.3.6
 # ./configure
 # make
 # make install
 # make install-man

Make the SHADOW web server home directory otherwise known as the DocumentRoot:

#mkdir /home/httpd/html

and copy the SHADOW home page image:

 # cp cid-1.5/httpd/home/shadow.html /home/httpd/html

Make the directory to store SHADOW html output files:

 # mkdir /home/httpd/html/tcpdump_results

Recursively copy the SHADOW images to the /home/httpd/html/images directory:

 # cp -R cid-1.5/httpd/images* /home/httpd/html

Copy the SHADOW cgi-bin files to the Apache cgi-bin directory.

 # cp /usr/local/logger//cid-1.5/httpd/cgi-bin/* /usr/local/Apache/cgi-bin

Review and modify each cgi file to reflect the names of your sites.

After all files have been copied, allow the non-privileged account that you previously created to become the owner of all web server files. The user whose processes will fetch the raw files and run the SHADOW scripts, probably not root for safety, should own these files. In our case, these files would be owned by the non-privileged sans-analyst account.

 # chown -r sans-analyst /usr/local/Apache
 # chown -r sans-analyst /home/httpd/html

Review the /usr/local/apache/conf/httpd.conf file and make sure that the variables are configured as follows:

ServerRoot /usr/local/apache
User {your non-privileged account}
Group {same as your non-privileged account}
ServerAdmin {your e-mail address}
ServerName {the hostname of your analysis system}
DocumentRoot /home/httpd/html [note: changed from /usr/local/apache/htdocs]
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
<Directory /home/httpd/html > [note: changed from /usr/local/apache/htdocs]
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>
Order allow,deny
Allow from 198.x.x.x
Allow from mypc.domain.org
Allow from yourpc.domain.org [note: restrict the Allow from entries to your local analysts]

Note:
Some systems require capital r for the recursive chown option.

After you have edited the httpd.conf file, you can start the Apache web server and test it by using a web browser (make sure that you use a system that is in your Allow from directive. To start Apache:

 # /usr/local/apache/bin/apachectl start

Add the Apache Web Server startup command to your UNIX system startup files, typically in /etc/rc.d/rc.local or /etc/rc.d/init.d.

Action 4.6 Configure the SHADOW analyzer

Make sure that the /LOG directory for the raw tcpdump files is owned by the non-privileged user account. In our case, the non-privileged sans-analyst user account on our analyst system would own these files. In addition, create subdirectories under /LOG for each site. For example, if sensor systems will be located at each of the three sites named Site1, Site2, and Site3, then create three directories: /LOG/Site1, /LOG/Site2, and /LOG/Site3.

The analyst system uses site-unique Perl header files located in the /usr/local/logger/sites directory. A default configuration file called GENERIC.ph is also located in that directory and should be copied to another file with the name of the site. For example, if you have three sites called Site1, Site2, and Site3, then the GENERIC.ph file should be copied to Site1.ph, Site2.ph and Site3.ph in the /usr/local/logger/sites directory. The contents of GENERIC.ph are:

```
# Variables needed by the analyzer scripts. Tailor this file
# to define the paths for different sensor sites.
#
$CID_PATH =  /usr/local/logger ;
$ENV{PATH} =  /bin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:$CID_PATH ;
#use POSIX qw(strftime);
use Time::Local;
#$SITE= SITE1 ;
@SITE_IP=( 172.16.31 ,  192.168 );
$SENSOR= sensor01.goodguys.com ;
$WEB_SERVER= www.goodguys.com ;
$SENSOR_DIR= /LOG ;
$ANALYZER_DIR= /LOG/$SITE ;
$OUTPUT_WEB_DIR= /home/httpd/html/tcpdump_results/$SITE ;
$URL_OUTPUT_DIR= /tcpdump_results/$SITE ;
$FILTER_DIR= $CID_PATH/filters/$SITE ;
$SCAN_THRESHHOLD =  5 ;
$LEVEL2_HOST= second.goodguys.com ;
$LOG_FILE =  /tmp/${0} ;
#
# Set the time for consolidate.pl to 3 days and 15 minutes ago.
#
$CONSOLIDATE_TIME=time-3*24*60*60-15*60;
#
# Set the date for cleanup.pl to 2 days + 15 minutes ago.
#
$CLEAN_DATE=strftime( %y%m%d , localtime(time-2*24*60*60-15*60))
```

Each of the variables in this file should be configured to fit the site at which the SHADOW system is being installed. The definitions of each term are as follows:

- ¥ $CID_PATH: the location of the installed SHADOW software: default: /usr/local/logger.
- ¥ $SITE: the name of the site, used as a subdirectory for storing both raw tcpdump files and output html files. The site name is a required parameter on the various SHADOW scripts.
- ¥ $SITE_IP: IP addresses with the purview of the site. Not used yet.

- ¥ $SENSOR: the official name of the sensor that this analyzer is to fetch data from.
- ¥ $WEB_SERVER: the official name of the computer running a web server to display the analyzer data. (Currently the official name of the analyzer.)
- ¥ $SENSOR_DIR and $ANALYZER_DIR directories on the sensor and analyzer in which the raw tcpdump files are stored.
- ¥ $OUTPUT_WEB_DIR and $URL_OUTPUT_DIR absolute and relative paths for the location of the analyzer output html files.
- ¥ $FILTER_DIR: the path to the location of tcpdump filters for this site, default filters subdirectory of SHADOW distribution.
- ¥ $SCAN_THRESHHOLD: the number of different systems within your site that can be scanned by an outsider before being flagged as a scanner on the analyzer output.
- ¥ $LEVEL2_HOST: the name of a system to which the data is shipped for secondary analysis. (Not currently used.)
- ¥ $LOG_FILE: the name of a file that contains debugging information for the SHADOW scripts. (Currently set to the name of the script.log in the /tmp directory.)
- ¥ $CONSOLIDATE_TIME: the amount of time after which the hourly raw tcpdump files are consolidated into daily files by the consolidate.pl script.
- ¥ $CLEAN_DATE: the amount of time after which the hourly raw tcpdump files are removed from the sensor s disks.

Once the Perl header files in the /usr/local/logger/sites directory have been modified, the SHADOW filter subdirectories must be configured. Create subdirectories under /usr/local/logger/filters for each site. For example, if sensor systems will be located at each of the three sites named Site1, Site2, and Site3, then create three directories:

 # mkdir /usr/local/logger/filters/Site1 /usr/local/logger/filters/Site2 /usr/local/logger/Site3

Copy the template filters (which will need to be customized for your site). As an example using our three site example:

 # cp /usr/local/logger/cid-1.5/filters/TEMPLATE/* /usr/local/logger/filters/Site1

 # cp /usr/local/logger/cid-1.5/filters/TEMPLATE/* /usr/local/logger/filters/Site2

 # cp /usr/local/logger/cid-1.5/filters/TEMPLATE/* /usr/local/logger/filters/Site3

Edit the filter files, ip.filter, tcp.filter, udp.filter, and icmp.filter in each of the Site directories. All the 192.168 and 172.16 IP addresses should be changed to reflect the IP addresses of machines at your site. As an example of tcpdump filters, the following line is included in /usr/local/logger/filters/TEMPLATE/tcp.filter:

tcp and

...

((tcp[13] & 2 != 0))

The first tcp in the example above means match TCP packets (as opposed to UDP, ICMP, IGRP, IGMP etc). The tcp[13] refers to the 13th byte of the tcp header. This is the location of the TCP CODE BITs field. The & 2 != 0 is how we mask this field to test for SYN. This causes all TCP packets with the SYN flag set to be written to a file. Several reviewers felt that just SYN and FIN packets could be used to define TCP connections. This could be accomplished by: tcp and (tcp[13] & 3 != 0).

TCP CODE BITS

Often called flags, the six bits in this field are used to interpret how to handle a TCP packet. They are:

URG Urgent pointer field is valid

ACK Acknowledgement field

PSH Push requested

RST Reset (break) connection

SYN Synchronize (start) connection

FIN Finished, all done

Action 4.7 Test fetchem.pl.

From the non-privileged account on the analysis system, run fetchem.pl interactively:

$ /usr/local/logger/fetchem.pl -l {site} -debug

and see if the files are correctly being copied from the sensor into the /LOG directory. Verify that output html files are being written into the /home/httpd/html/tcpdump_results/{SITE}/ directory structure. By adding the parameter -debug to the fetchem.pl call, debugging information is written to the /tmp/fetchem.log file. The script should be run as a regular user.

Common problems are file and directory permissions.

Action 4.8 Edit cron to call fetchem.pl

A file called analyzer_crontab in located in the /usr/local/logger directory and is a template for the crontab to be run on the analyzer:

```
# Crontab for a tcpdump Analyzer. Ensure system clock is consistent.
# If this system is not on the Internet, you ll need a standard time source.
#
# These two entries must be done as root.
#
17 23 * * * /usr/bin/rdate -s time-a.nist.gov
18 23 * * * /sbin/hwclock --systohc
#
# The following entries need not be done as root.
#
5 * * * * /usr/local/logger/fetchem.pl -l SITE
17 1 * * * /usr/local/logger/cleanup.pl -n SITE
25 0 * * * /usr/local/logger/consolidate.pl -l SITE
```

Step 5 Testing the Display System

Action 5.1 Introduction to the display system

Fetchem.pl s job is to pull the raw tcpdump file from the sensor machine, store it in a date descriptive directory on the analyzer, and run it through tcpdump again with a set of filter files which will detect those events you suspect may be intrusion attempts. The output from this run is written to a file within the document directory of an http web server. People responsible for watching the network simply use their web browser to the analysis system to look at the web pages containing events flagged by the events filters. The web files are chained together by forward and back arrows, so the analyst can go from hour to hour easily. Dir-it is the program that creates and updates the home page that provides an index of all the data. The dir-it program was written by Robert Niles and we have included it in the source distribution. Consequently, pointing a web browser to this domain allows an analyst to quickly review any findings of the tcpdump filter run.

Action 5.2 Testing the display system

Testing is simple: point your web browser to the analysis station: http://analysthost.org. The resulting page should be labeled SHADOW and should have a push button labeled Tool Window . Below are some troubleshooting tips:

(a) Make sure that the httpd processes are running

$ ps ax | grep httpd

(b) For clues , check the error_log file and the access_log files in the /user/local/apache/logs directory on the analysis system. Try:

$ tail -f /usr/local/apache/logs/error_log

(c) If you get a Forbidden - You don t have permission to access / on this server , then you may have a problem with your httpd.conf file in the /usr/local/apache/conf directory. Look specifically for the Allow from directive in the httpd.conf file.

(d) Common problems are file ownership and protection of the /home/httpd/html files and the /usr/local/apache files (especially in the cgi-bin subdirectory).

SECTION 4 Filtering Packets

Step-By-Step

The SHADOW analysis process is less concerned with the payload or contents of the packets than with the header information. Like a letter through the mail, SHADOW is concerned with the information on the outside of the envelope, not the letter inside.

The heart of SHADOW s analysis system is a program called `tcpdump` which supports two kinds of network analysis:

- ¥ It can collect raw packets
- ¥ It can filter packets looking for activity patterns

This section is a step-by-step guide for learning the language that controls tcpdump s filtering activities. The man page (documentation describing the program s operation and options) for `tcpdump` is an excellent resource, though its 18 pages can be daunting.

Since most sites maximize their detection results by customizing the filters, they need to know the language used to create filters for intrusion detection. This section teaches that language using examples. It begins with simple filters and progresses to more complex filters developed by combining simple filters. Finally, it shows a filter that searches for a long list of events one might wish to monitor. A sample set of filters is included with the SHADOW software so that the toolkit is ready to be customized right out of the box.

Remember, in the SHADOW architecture, the sensor reads raw packets from the network and saves them to a file. The analysis system reads those packet files and applies filters to look for specific patterns.

SHADOW's Logfile Naming Format

SHADOW file names have eight digits, with four pairs of digits created like this:
98 = The year, 1998 (we aren t doing date mathematics so Y2K isn t an issue)
01 = The month, January
21 = The day
16 = The hour in military time.

We roll the files over hourly. The files suffix is ‘.txt , and, being compressed, they often have the additional ‘.gz suffix. An example file name might be:
98012116.txt.gz

A Note for First Time Network Analysts

A neophyte intrusion detection analyst should be very slow to panic! IMAP, PORTMAP, and so forth are common networking attacks, but are also commonly used services. Take time to learn how your site does business, network-wise. Once you start looking, it is probable that you will detect some attempted attacks. Most systems that are kept up-to-date with security patches will not be affected by these attacks. Calm analysis is the best path to follow.

Simple Filters

Let s examine a set of simple, useful filters. These simple filters will later be combined with and , or , and so forth to create more complex filters.

The Packet Record

The first 96 bytes of a packet are available for header analysis. Tcpdump typically displays the packet in a human-readable format like this:

```
Timestamp    Source  Source Port  >    Destination.DSTPort  Protocol
00:00:05.327 example.org  1025    >  192.168.64.15.53         UDP
```

The timestamp tells when the packet arrived (with millisecond precision). The source site name (or IP address) is shown along with the source port number. The destination IP address and port are displayed along with the protocol. This standard format will be used throughout this document to show packet headers.

It is the goal of the rest of this section to show simple filters that will choose ‘interesting packets from the real-time data stream so that they can be stored in a log file and subsequently analyzed either mechanically or by a human.

Example 3.1: Simple Filter to Detect Telnet Packets

Busy networks see millions of packets per hour. Saving all their headers results in very large files, so the files are kept compressed. Here is the command to uncompress the binary log files and send the results to tcpdump, and at the same time filter the results for the desired packets:

```
$ gunzip -c yourlogfile.gz | tcpdump -r -  tcp and dst port 23
```

The ‘tcpdump -r - command runs tcpdump and instructs it to read from a file (the 1- is the 1file name and means 1standard input the result of the gunzip) instead of a network interface. The ‘filter is in quotes. Only those packets that satisfy the ‘filter requirement will be ‘tcpdump ed.

For the remainder of these filter examples, we will provide only the filter specification itself rather than the whole decompression command line.

This filter selects those packets that have protocol type ‘tcp and destination port number 23. Since the telnet protocol uses port 23, this filter selects telnet packets. These packets are then printed in human readable format to the standard output (usually redirected into a logfile). Later, programs and humans will analyze the packets in the logfile.

Example 3.2: A Filter for IMAP

Note:
The number in brackets after tcp "tcp[13]means the 14th byte (counting from 0) in the tcp header. The term "!=" means "not equal to." The ampersand "&" means mask bits starting from the right. Here "2 != 0" means the bit in the second position from the right of the tcp code bits which means the SYN flag is set.

The following filter scans packets for tcp SYN packets sent to destination port 143, the IMAP service, and a very common attack destination.

```
tcp and (tcp[13] & 2 != 0) and (dst port 143)
```

This filter above is pretty simple. The term dst port refers to the destination port. Most UNIX computers have a file called /etc/services which provide names for these numeric ports. For instance: email (smtp) is TCP port 25, telnet is TCP port 23 and so forth. IMAP, as we mentioned earlier, is port 143.

If the filter matches a target pattern in a data file, tcpdump displays the results in the record format described earlier. Here is an example from a file named 98012116.txt (which was created January 21, 1998, around 4 pm):

```
16:33:14.296403 linux.zsagvari.c3.hu.35486 > 204.34.256.1.imap: S 0:0(0) win 512
```

The 'S means the SYN flag was set. The next numbers are in the format

'first:last(nbytes) which describes the sequence number fields; see the tcpdump(1) man page for details

About SYN and FIN
Many sites block IMAP (TCP port 143) because a vulnerability in this service was one of the most successful attack exploits ever. If the service is blocked, all that will be detected is the connection request (active open), which is a packet with the SYN flag set and no ACK. One variety of IMAP exploit script has been detected in which the packets have bhot the SYN and FIN flags set. The purpose of this might be to avoid completing the TCP three-way handshake and therefore avoiding detection. By masking tcp[13] with "&3 != 0", you get both the SYN and FIN; if you use "& 2 != 0", you will get only SYN.

Example 3.3: An ICMP Filter

Here is a simple filter from the tcpdump man page that looks for all ICMP packets that are not echo requests or replies (i.e., not ping packets):

```
icmp and icmp[0] != 8 and icmp[0] != 0
```

ICMP message types are not listed in /etc/services. However, you may find /usr/include/netinet/ip_icmp.h to be helpful as it defines the names of the ICMP message numbers.

ICMP, the Internet Control Message Protocol, is not usually exploited to break in to your site s computer systems. It is, however, being used for numerous denial of service attacks. ICMP was designed as a network health indicator and if your machine were suddenly to exhibit a lot of TIMEXs (time exceeded), UNREACH (net, host, protocol ... unreachable), or SOURCEQUENCH indicators, this could imply network problems.

Time exceeded and port unreachable messages are also a potential result of running the traceroute program from a host computer at your site. Traceroute sends out packets to probe for the identities of all the routers along a network path and gathers the data it needs from the ICMP messages.

Example 3.4: A Filter to Detect Broadcasts

ip and ip[19] = 0xff

One of the classic attacks with ICMP, in which very large ping packets are fragmented, is often combined with a broadcast. After all, why go for a system when it is just as easy to attack an entire subnet? Many automated scanners also send probes to broadcast addresses; this filter will detect these as well. A filter like the one above will detect broadcast ICMPs and any other broadcasts.

Here is sample output from a filter match for ICMP:

16:00:03.828071 fraggg.org > 128.256.15.255: (frag 27392:548@1480)
16:00:03.896593 fraggg.org > 128.256.1.255: (frag 21248:548@1480)
16:00:06.118729 fraggg.org > 128.256.15.255: icmp: echo request (frag 52480:1480@0+)
16:00:06.250349 fraggg.org > 128.256.15.255: (frag 52480:548@1480)

Directed Broadcast

If this type of broadcast appears on your network, it is likely that your router (or your Internet Service Provider's router) is configured to allow the "directed broadcast" option. We recommend that you disable this via the router configuration. For further information please see http://www.quadrunner.com/~chuegen/smurf.txt .

Fraggg.org could well be spoofed. When you send an ICMP echo request (ping) to a subnet broadcast address (e.g., 172.16.1.255), all the systems on 172.16.1 are supposed to answer the computer that pinged them. If many subnet addresses are pinged and the source address is spoofed, the ensuing barrage of ICMP echo replies can flood fraggg.org s Internet connection, thus effecting an attack on fraggg.org by your organization!

Example 3.5: Land Attack Filter

Some computer systems freeze if they receive a packet in which the source address is spoofed to be the same as the destination address, another denial of service attack. Here is a filter to detect this:

ip and ip[12:4] = ip[16:4]

Example 3.6: A Filter Designed to Detect Hostile SNMP

(udp port 161 or udp port 162) and not src net 172.17

SNMP was developed for network management. However, adversaries can use SNMP to collect information about your networks and computer systems. Network appliances (such as routers, hubs and bridges) often have SNMP agents built in and, in addition, many other devices (such as print servers and X terminals) also have built-in SNMP agents. These devices use a community string to control access. Many SNMP agents default to the community string public , which means just that.

The example above shows how to provide an exception address. If the source network is 172.17.xxx.xxx, the packet will not be matched. It is common for a site to allow connections of a certain type from another trusted network, but not to allow them in general.

Example 3.7: A Filter to Watch for the Berkeley r-utilities

ip and (tcp dst port 512 or tcp dst port 513 or tcp dst port 514)

The r-utilities (rlogin, rcp, rshell and so forth) enable two trusted systems to exchange files and commands without authentication. Ideally, systems that need to do this type of trusted exchange across the Internet will change to Secure Shell or some other more secure mechanism. Two problems arise with the use of r-utilities: (1) /etc/hosts.equiv and (2) individual user s .rhosts files. If a system has a + + in its host.equiv (too often supplied by vendors), that means it trusts all users from all systems. Needless to say, it is wise to look for packets with r-utilities from unknown sites.

This filter might be a good candidate to exclude an exception address.

Example 3.8: A Filter to Detect Access to Portmapper

ip and dst port 111

Many /etc/services files call this port sunrpc . RPCs tend to connect to the portmapper at either TCP or UDP 111 to find other services. This is a very old (and effective) gateway to a series of attacks. In late 1997, the SHADOW team started to see an increase in portmap attempts.

From 98012403.txt:
03:20:42.579548 netgate.srn.com.829 > ns2.nnnn.navy.mil.sunrpc: S 3648872793:3648872793(0) win 512 <mss 1460>
03:20:45.547040 netgate.srn.com.829 > ns2.nnnn.navy.mil.sunrpc: S 3648872793:3648872793(0) win 31744 <mss 1460>

Note:
The 'S', or SYN packet flag. A firewall screens the DNS server ns2.nswc.navy.mil so that the sunrpc attempt never actually reaches it.

Example 3.9: An NFS Filter

NFS is a good service to keep an eye on, and here is a filter to do it:
ip and udp port 2049

05:17:50.562188 j.K.EDU.885 > dorad.nnnn.navy.mil.nfs: 40 null
05:17:52.553265 j.K.EDU.885 > dorad.nnnn.navy.mil.nfs: 40 null
05:17:56.551772 j.K.EDU.885 > dorad.nnnn.navy.mil.nfs: 40 null

Note:
Port 2049 might be NFS or it might be used for something else. Don't panic the first time you run this!

Not all hits are intrusion attempts. Our assessment of the data series above is that it shows the results from an automated process with a typographical error that happens to be our server dorad s Internet address. Since NFS can also run on TCP, it might be wise to modify the filter to say:
ip and udp port 2049 or ((tcp[13] & 2 != 0) and (dst port 2049))

or
ip and udp port 2049 or tcp[2:2] = 2049

Example 3.10: A NetBIOS Filter

Microsoft Windows For Workgroup, Windows 95, Windows NT and SAMBA all use a protocol called NetBIOS to communicate over the Internet. This protocol uses ports 137, 138, 139 of both TCP and UDP. Here is a step by step construction of this filter, one part at a time:

ip and ...

will match both TCP and UDP. Now to match the port numbers 137, 138, and 139:

... port 137 or port 138 or port 139

Add parentheses to ensure correct parsing precedence:

ip and (port 137 or port 138 or port 139)

If the destination is NetBIOS nameservice (port 137) and source port is not using port 137, the connecting computer is probably SAMBA. Under most circumstances you probably don t care, but if you do:

ip and dst port 137 and not src port 137

Example 3.11: An X11 Filter

X11 is a protocol that enables a program run on one UNIX host to display the results in a window on a potentially different X-enabled workstation. If configured incorrectly, it is possible for an attacker to obtain a screen dump of what is currently shown on your workstation, monitor all of your keystrokes, or even insert their own keystrokes into some software that you may be running. X11 uses TCP port 6000 and sometimes the nearby higher ports. An example filter for this would be:

tcp and (port 6000 or port 6001 or port 6002)

or, perhaps better this way:

tcp and (tcp[13] & 2 != 0) and (dst port 6000 or dst port 6001 or dst port 6002)

which detects packets with the SYN (Synchronize or open a connection) to these ports.

Example 3.12: An IRC Filter

IRC is a real-time chat system that runs over the Internet, usually using TCP port 6667. When an attacker compromises a UNIX system, it is very common for the attacker to install an IRC server on that computer. Sometimes they will then use the server to chat with their cohorts about other attacks. So, even if your organization makes absolutely no use of IRC, it is useful to monitor for it since it can be a warning sign of a compromised machine. A filter for this is:

tcp and port 6667

Example 3.13: An NNTP Filter

NNTP is the protocol used for exchanging the messages that appear in Usenet newsgroups. If you have an NNTP server on your network that communicates with an external server (often the NNTP server of your Internet Service Provider), you will undoubtedly see a huge amount of traffic for its TCP port number, 119. If you do not exchange messages with the outside world via NNTP, you might still see occasional packets, possibly from attackers that want to break into your server, but possibly also from attackers seeking sensitive information from your private internal newsgroups.

A filter for NNTP is:

```
tcp and port 119
```

Example 3.14: A Filter to Detect Fragments

Fragmentation is sometimes used to evade intrusion detection systems. Excessive fragmentation can also indicate problems with network configuration. Sites that utilize NFS over the Internet probably do not want to run this filter as NFS likes large data chunks and can create many fragments.

A filter to detect fragments is:

```
ip[6:2] & 0x2000 != 0
```

Example 3.15: A Filter to Detect Socks

The Socks security protocol runs on TCP port 1080, to which the SHADOW team has detected several probes. One CERT advises that there are exploits that enable an attacker to establish a connection to port 1080 and then bounce out from the system to attack or probe another computer. This way, the attack looks to other intrusion detection systems like it comes from your network. It is important to detect SYN attempts to 1080 since it is a commonly used port for FTP and HTTP port counting.

A filter to detect Socks is:

```
tcp and (tcp[13] & 2 != 0) and (dst port 1080)
```

Example 3.16: A “bad events” Filter

Sophisticated filters can be constructed to scan for any set of events you want to detect. For example, here s a portion of a script called bad_events that is used to detect any packets that could indicate suspicious activity that might warrant further attention.

In the next section, we provide a complete set of all the filter atoms the SHADOW team was using the last time the code was frozen. The purpose for the example filter shown here is to illustrate how the atomic filters with which we are gaining experience can be strung together with and s and or s.

If the filter syntax appears to be a bit tricky, try a book called *Internetworking with TCP/IP*, Volume I by Douglas E. Comer, before trying anything really fancy. This book is a good investment if you are attempting to become an intrusion detection analyst for your organization.

TCP/IP Illustrated, Volume 1 by Richard Stevens is another excellent resource that gives examples using tcpdump output. These books are popular, so you may be able to borrow one until your copy comes in.

Here is an example filter that strings together all the atomic filters.

```
( tcp
and
(tcp[13] & 3 != 0)
and
( (dst port 143)
or
(dst port 111)
or
(tcp[13] & 3 != 0 and tcp[13] & 0x10 = 0 and dst net 172.16 and dst port 1080)
or
(dst port 512 or dst port 513 or dst port 514)
or
((ip[19] = 0xff) and not (net 172.16/16 or net 192.168/16))
or
(ip[12:4] = ip[16:4])
)
)
or
( not tcp
and
not igrp
and
not
dst port 520
and
( (dst port 111)
or
(udp port 2049)
or
((ip[19] = 0xff) and not (net 172.16/16 or net 192.168/16))
or
(ip[12:4] = ip[16:4])
)
)
```

Example 3.17: Putting the Pieces Together

Now that we have reviewed the syntax of the individual filter pieces and seen an example of a composite bad events filter, let s take a closer look at building a real system to use for automatic network monitoring.

The first problem we have to address is that several of the tcp filters above will yield a lot of false alarms if users on your network are doing http, ftp, or exchanging email with the outside world and these are normal activities!

For example, the Socks filter looks for tcp packets having either source or destination port 1080 where the syn flag is set. If the composite filter given in the previous example is used, the filter will grab tcp packets involving port 1080 where either (or both) the syn and fin flags are set. Usually, we are interested in active open connections, i.e., the first connection in the tcp 3-way handshake. If we use the filters as given above, we will give false alarms on passive open tcp connections.

Imagine a user on your network accesses a web page on some remote server. They connect (the active open) to port 80 on the webserver and use a source port of 1080. When the webserver responds back acknowledging the syn it received and sending its own syn (the passive open), it will connect to the client machine on port 1080 and use a source port of 80. Notice that the passive open connection will meet the criteria of the filter we have discussed (the destination port is 1080 and the syn flag is set). However, this connection is clearly not the result of an attacker trying to execute an exploit on tcp port 1080. Further, if we do not account for passive open connections, any attempt to look for unknown tcp port accesses (by defining what is known and looking for the rest) will be foiled by false alarms as the source port for an active open is typically chosen at random.

The solution to this problem is simple: we look for tcp packets where the syn flag is set but the ack flag is not:
tcp and (tcp[13] & 2 != 0) and (tcp[13] & 0x10 = 0)

and note that this filter will also pick up connections where the syn and fin flags are set simultaneously.

If we wanted to write a filter that specifically looks for tcp connections where the syn and the fin flags are set simultaneously, we would use the following:
tcp and (tcp[1 3] &3 = 3)

In fact, the SHADOW team has recently seen connections coming to our networks having the syn, fin, ack, push, reset, and urgent flags all set at the same time. If you are looking for new attack patterns, it could be educational to screen for inbound packets having weird combinations of these flags set.

For the system to be effective as a whole, it is necessary to enumerate all possible events of interest for which one might screen. These components can be logically grouped together to yield a sophisticated filtering capability.

At the SHADOW site, we build several filters up from these components and then use a perl loop to call the filters in sequence since tcpdump can only handle a certain number of filters in a single invocation. Of course, some of the filters may overlap and a given packet might be extracted from the data by more than one filter, so the loop is followed by a call to a perl program that sorts and groups the filter outputs, throwing away duplicate connections. The filters we have built are available at the website, as are perl programs we use in concert with the tcpdump output to provide more advanced capability. A subsequent section discusses some advanced techniques.

When You Get a New Pattern

The SHADOW team archives a growing library of patterns. We hope that SHADOW users everywhere will help us to continue this work. If you see a unique and unusual pattern with your SHADOW system, please sanitize the source and destination addresses and mail the pattern and what you have been able to determine about it to shadow@nswc.navy.mil. Please obfuscate the source and destination addresses first!

SECTION 5 Analyzing Filtered Packets

Display the Information for Maximum Analytical value

Let s consider the filter system described so far. the sensor in the DMZ is collecting data a lot of data. You have examined this data with filters.

When the files get very large, it takes increasingly large amounts of time to parse the files with filters. Tasks that take time and effort motivate many people to quit performing those tasks. For this reason, information display is every bit as important as filtering.

One of the advantages of the SHADOW architecture is that a single analysis system can support multiple sensors (but make sure you stagger the file downloads in the cron files so that you aren t filtering the sensor data from multiple sensors at the same time, of course).

Introduction to the Display System

The fetchem.pl perl script writes the information from filter matches in html format into a directory that is served by the web server. Those responsible for watching the network use their web browser to see what is going on. The web files with events flagged by the events filters are chained by forward and back arrows, so the analyst can easily go from hour to hour (i.e., from data file to data file).

Robert Niles dir-it is a program that creates and updates the home page that indexes all the data; it is included in the SHADOW toolkit source distribution.

There are many web servers available. The one that we have used for displaying intrusion detection information for several years is the popular freeware Apache web server. It is available from http://www.apache.org.

Using a “bad_events” Filter to Survey Your Network

Intrusions are detected exactly one way. By one technique or another, they are isolated from the traffic stream and found to be interesting (abnormal), non-business related traffic. The easy detects in intrusion detection for a traffic analysis approach are the services that could not possibly be part of your organization s normal business related traffic.

One great way to start is to locate your organization s firewall policy. This way you can determine the things your organization has decided to allow. Remember the great axiom of firewall policy:

That which is not specifically allowed is prohibited.

Therefore all TCP and UDP ports below 1024 that are not allowed should be flagged with the bad events filter. All TCP ports above 1024 that are not allowed can be flagged if there is a SYN packet to that port and all UDP ports above 1024 to ports that are not allowed should be flagged.

If your organization does not have a firewall (or the firewall is extremely permissive), you will need to write a set of filters that will let you examine all traffic and then determine statistically which ports are on normal business traffic and then monitor for exceptions. For TCP:
((tcp and (tcp[13] & 2 != 0) and dst net 192.168 and tcp[2:2] < 1024) ...

This filter will list all SYN packets or attempts to open a connection to a port below 1024 for the destination network 192.168 (put your network here, of course). If you want to see all incoming connections, simply modify the filter to say:
((tcp and (tcp[13] & 2 != 0) and dst net 192.168) ...

Beginning Intrusion Detection

Now that you have completed surveying your network traffic and have created a bad_events filter to satisfy your needs, you can begin using this toolkit for intrusion detection.

The crontab-initiated tcpdump filters will place the files of interesting events in a directory arranged by hour. As you use the web browser to click on each hour, you will see the events that were flagged during that time period.

If you want further information on a particular event, you may want to run one of the analysis scripts supplied with SHADOW. For example, one of the most common and useful tools is the one_day_pat.pl program. Provide it with a date and a pattern and it will check all the hourly files for the pattern over the entire day specified and output this information appropriately.

You can run one_day_pat.pl through the web GUI or from a command prompt shell. If you run it from the web interface, the date (-d) and the sensor or logger (-l) information is filled in automatically. If you choose to run one_day_pat.pl from the command prompt, you will need to supply these command line options, or the software will make a guess which might or might not be correct. The additional option -n avoids name lookups and speeds operations substantially.

For instance, if you check the hourly files and see an entry concerning srn.badguy.org and want more information about all actions involving srn.badguy.org for the day, try:

$ one_day_pat.pl -d 980520 -l NNNN -p 'host srn.badguy.org'

TCP and UDP services can be partitioned into two categories, those ports below 1024 and those above. The reasons for this are rapidly becoming historical. On UNIX systems, root level access was required to operate a service below port 1024. The idea was that such a service could be trusted since anyone with root access was presumably trustworthy. Further, the ports below 1024 are fairly well defined and used consistently (i.e., TCP port 25 is very likely to be sendmail; 23 telnet; and so on).

Port numbers above 1024 are pretty dicey. For instance, TCP port 1080 may be for Socks, but if you want to detect a probe on Socks you will need to look for a SYN packet (active open) to TCP 1080, otherwise you will detect a lot of perfectly normal FTP and HTTP file transfers.

In a file transfer, both the source port and the destination port generally count up. In other words, for each packet that is transferred, both the source and destination increment by 1. So if a file transfer started at destination port 1060, the next packet would go to 1061, then 1062 and soon pass 1080. When it passed 1080, it would match the filter, unless you had set the filter to look for an active open, showing a connection beginning at 1080.

and all traffic for that day to or from srn.badguy.org will be displayed. This script is also helpful to test for events that are not in your bad_events pattern.

The web page only displays the events matched by the bad_events filter or the additional perl scripts. However, Networks are dynamic and it is wise to constantly monitor for change. From time to time, we recommend that you try experimental filters to see what you can detect. As an example, when the Advanced Scanning Techniques (CERT Incident Note IN-98.04) were first detected, they did not match any of the atomic filters in the bad_events pattern. This means they were not displayed. However, some of the probers sent packets at a rate of more than seven different target hosts per hour, which is where we had our counter set on our analysis system. So what was displayed on the bottom of the page were probing hosts which were flagged by the scan director. To find out what they were doing we used one_day_pat.pl with their name:

$ one_day_pat.pl -d 980520 -l NNNN -p 'host hook'

We got back a bunch of TCP resets to different hosts, but no other traffic:

17:40:45.870769 hook.24408 > target1.1457: R 0:0(0) ack 674719802 win 0
17:40:53.025203 hook.33174 > target2.1457: R 0:0(0) ack 674719802 win 0
17:41:12.115554 hook.36250 > target3.1979: R 0:0(0) ack 674719802 win 0
17:43:37.605127 router > hook: icmp: time exceeded in-transit
17:43:43.139158 hook.44922 > target4.1496: R 0:0(0) ack 674719802 win 0

The filter set in bad_events will need to be updated often. New services and new probes and attacks are being developed all the time. Be willing to construct filters and run them on the data, just to see what they will do. You may discover a previously unknown attack technique.

Please also note SHADOW 1.4b (and later versions) has a specialized version of one_day_pat.pl that tests all sensors for twenty four hours and can find very stealthy probes. Look4scans.pl uses filters that are located in /usr/local/logger/filters/generic which are the non-site dependent filters. To use look4scans:

$ look4scans.pl -d 981003 -f filter_name_of_a_generic_filter

For instance when we were trying to gather more information about probers who were using odd combinations of TCP code bits, we used the generic filter multiflags which is shown below:

```
tcp and (not src port 80)
  and (tcp[13] & 0xf != 0)
  and (tcp[13] & 0xf != 1)
  and (tcp[13] & 0xf != 2)
  and (tcp[13] & 0xf != 4)
  and (tcp[13] & 0xf != 8)
  and (tcp[13] & 0xf != 9)
```

When we ran:

$ look4scans.pl -d 981003 -f multiflags

We discovered the information that lead to this:

09:42:16.839047 prober.30975 > CONDOR.60: SFRP 2029977660:2029977691(31) ack 2029977660 win 60 urg 60 <[bad opt]> (DF)

11:08:51.046434 prober.14690 > ROBIN.12916: S 962736756:962736805(49) win 12916 urg 12916 (DF)

11:36:59.781247 prober.30974 > CONDOR.x400-snd: SRP 2029912168:2029912244(76) ack 2029912168 win 104 urg 104 <[bad opt]> (DF)

12:37:46.893338 prober.30973 > EAGLE.64: FRP 2029846592:2029846625(33) ack 2029846592 win 64 urg 64 <[bad opt]> (DF)

The machine named Prober, with no stimulus or other traffic, is sending odd TCP header combinations to core (DNS/MAIL) servers. We also see this is occurring at a slow (<2 packets/hour) scan rate. The purpose is apparently to test for various TCP stack responses, perhaps in order to determine the operating system of the target.

Trying experimental filters can also help revise assumptions by providing additional information. For months when we saw large numbers of icmp echo requests we felt they were either an attempt to execute a denial of service (Smurf) or to develop a network map (Pingsweep or Pingmap).

One day Vicki Irwin, a SHADOW analyst, suggested we test for size. Normally, the ip header would be 20 bytes (if no options are set), the icmp header on an icmp echo request is 8 bytes, and the standard size for the data payload on an icmp echo request is 56 bytes. So if the total length of the ip datagram carrying the echo request is > 20 + 8 + 56 = 84 bytes then that could be an indication that something is up.

The decimal number 84 is 00000000 01010100 in 2-byte binary format, so in hex that would be 0x0054. And then the filter is:

icmp and (icmp[0] = 8) and ip[2:2] > 0x0054

When we ran this filter against the data, a lot of the Pingsweeps matched. Why would anyone send large packets to do mapping, or could it be denial of service after all? And how big are those pesky pings?

500 in hex is 01f4, so how about:

icmp and (icmp[0] = 8) and ip[2:2] > 0x01f4

And they still matched. We could try 1000 bytes:

icmp and (icmp[0] = 8) and ip[2:2] > 0x03e8

You guessed it, they still matched. For further information about how you can use large ICMP packets, please take a look at p. 152 in Stevens *TCP/IP Illustrated Volume 1*. He gives an example of someone attempting to use pings to determine the MTU of a dialup slip link. In this case the MTU is 552 and Stevens sends pings at 500 and 600 bytes to figure out what is going on.

We hope this example illustrates how one can use experimental filters to detect things that would otherwise be missed.

It is an essential task of intrusion detection to observe your site and learn its quirks. You must be able to detect the smallest deviance from the ordinary. The examples given above are only a few of the many types of attacks or probes you might see as you track your network data.

Although you may have a lot of data to examine, your effectiveness depends upon your being careful enough to observe the indications of an attack and be able to spot such an instance even if it is hiding in the middle of a huge amount of data. Again, this is where the one_day_pat.pl may become quite useful. Remember: attackers are persistent and are always trying new methods to gain access to your systems. It is vital that you know what constitutes standard traffic on your system and be sensitive to anything unusual.

Responding to a Detect: What if You Find Something?

If you are checking your hourly files and find something that looks suspicious but not really scary, the next logical step is to run one_day_pat.pl as described above to get more information. It is important to keep an accurate audit trail of everything that you discover in case the information is needed later to support recovery, or even prosecution. A better way to run one_day_pat.pl is to collect its results into a file. This can be done as follows:

$ one_day_pat.pl -d 980520 -l NNNN -p 'host srn.badguy.org' > host.time

then you can run

$ tail -f host.time

to view the results in real time as the command runs. The advantage of adopting this approach is that you will have a record of every action that you took. One helpful option you may want to include when using one_day_pat.pl is the -n option. This option causes the results to be displayed as only IP addresses, without wasting time matching IP addresses with names. If you are only concerned with IP addresses, this option may be useful to you.

If you are in a hurry, because what you saw on your screen was fairly scary, then you may want to run tcpdump directly on the hourly file (without name lookups using -n) for faster results. The following command will accomplish this:

$ tcpdump -n -r 98051512 'host 192.168.1.1' > host.time
$ tail -f host.time &

Whether you run one_day_pat.pl or tcpdump directly on an hourly file, now you can examine the data and decide whether you have a problem.

There are several things you might notice. If the packets are only incoming without replies, then your firewall and system protections are probably effective. If there are replies to the sending host, but they are ICMP and they say port unreachable , then again you might be OK. This is not great, though, since it is an indicator that either the packets are getting through your firewall or the external interface of your firewall has knowledge of your internal network.

If you see other replies, you probably want to alert your organization s computer incident response team. It is a good idea to set up some sort of reporting system in which the details of an incident are recorded for future use; you never know when this information will be useful, whether it be evidence for prosecution or a record for comparison with future incidents. For further information on how to establish an incident handling team or how to respond to incidents, see the SANS Computer Security Incident Handling Step-by-Step Guide (http://www.sans.org).

Many sensors have enough power that you can run an additional session of tcpdump with a filter to watch the attacker. There have been cases where an attack has come from several systems on a network, so you may want to collect information on the entire net. The following command sequence illustrates this:

```
$ ssh -l root sensor
# df
# cd /LOG
# mkdir host.date; cd host.date
# tcpdump -n 'net 192.168.1' > time &
# tail -f time &
```

This way you will have information to provide the incident handling team. Don t forget to stop the tcpdump process soon or later.

We strongly advise that you test this procedure before an incident. It is recommended that you establish a Secure Shell key relationship with your incident handling team in advance of an incident so that you can transfer incident information in an encrypted manner to protect against any potential network sniffers.

Enhancing your Intrusion Detection Capability

Intrusion detection is much more than just looking for a few bad events. How can we improve the capability of the system?

Sort the Data for Display and Resolve Hostnames from IP addresses

In order to display the traffic culled by the filters you have constructed, it is useful to sort the traffic in some meaningful way. Tcpdump will present the traffic to you sorted chronologically. If you are looking for scans or single line connections amidst a torrent of denial-of-service ping broadcasts (for example), you must sort and group the traffic appropriately to make subtleties stand out. Our current solution is to group the connections by source IP address and insert a blank line in the display between groups of connections originating from the same source. In this way scans become immediately obvious, and very noisy/overwhelming attack connections are clustered together and set aside so that the analyst can see the more subtle connection attempts.

The first step in this process is to run tcpdump with the -n option set. This option setting tells tcpdump not to resolve the hostnames, but rather to leave the IP addresses in numerical dot notation. The resulting traffic may then be parsed and sorted by source IP and then by time. As the traffic is written to the output, we check for and discard duplicate lines in the sorted data. When the source IP changes we insert a blank line. The script we use to do these tasks is available from the NSWC website.

Being human, we would prefer to read host names as opposed to IP addresses. Thus we sometimes run another script that reads the sorted output, resolves the host names from the IP addresses and re-insert them as tcpdump would have. A short perl snippet that performs the name resolution is given below.

```
#!/usr/local/bin/perl
$ip  = $ARGV[0];
@octets = split(/\./, $ip);
$binary_ip = pack "c4", $octets[0], $octets[1], $octets[2], $octets[3];
hostinfo = gethostbyaddr($binary_ip, 2);
$hostname = $hostinfo[0];
print $ip, "  ",$hostname,"\n";
```

This snippet should be executed with a single command line argument that is the IP address in standard 255.255.255.255 type notation. The resulting hostname is printed to the screen provided the DNS server is able to resolve the IP address.

Looking for Host Scans Independent of Protocol or Attack Type

One way to detect unusual activity on a network is to search for source IP addresses that are attempting to connect with many destination hosts over a given time period. We can find such hosts of interest independent of any particular protocol or attack type; in fact, looking for scans in this manner is a good way to detect new attack patterns.

For example, the SHADOW team has recently been observing scans of our network using tcp-reset packets. It is both time-consuming and uninteresting to monitor all reset packets sent to all hosts on a network, as this is a normal occurrence in tcp communication. However, these connections become more interesting when one observes a single host from the outside world sending reset packets to many machines on your network when no other dialogue has taken place.

The best way to screen for these connections is to look for inbound reset connections where no other traffic between the source and destination hosts has occurred. However, such a program will have difficulty executing fast enough to perform hour-by-hour analysis and the attack type will be limited to resets only. The next best thing in terms of detecting this attack is to screen for one-to-many correlations between source and destination IP addresses. We have discovered that this is a great way to find new probing or mapping techniques.

Another application finds scans that are embedded in a great deal of normal traffic. For example, monitoring all udp 53 (domain name server) and udp 137 (netbios name service) connections would be overwhelming or impossible on high-traffic-volume networks. However, sites are commonly scanned on these ports where hackers are looking for nameservers or machines running netbios. A scan-finder as discussed above is able to extract these scans out of the noise.

How does one write a program to do such a thing that does not take infinite time to execute? For example, we might write a program to make a list of all the source IP addresses that are sending connections to our protected network (tcp, udp, icmp, ..., everything). Then, for each source IP in the list, we would make another list of all the destination hosts to which the source is connected.

Using this scheme, each time a new connection is considered, we must search the entire list of source IPs to determine whether the source has been previously added to the list. Then, if the source host is already listed, we must search the corresponding list of destination IPs to find whether the current destination host has already been recorded. In the beginning, while the lists are short, this process proceeds quickly. But, as the list of source addresses grows, the search becomes slower. Similarly, in the case of a large network scan, the list of destination addresses (corresponding to a given host) grows very large, thus slowing the search.

Our solution to this problem is given at the NSWC website. The basic idea is as follows: we first filter the raw data using tcp dump. We grab all inbound traffic except tcp connections having source port 80. We consider only inbound connections in order to reduce the amount of traffic that we have to process. We exclude tcp connections having source port 80 to avoid mistaking the connections back from a commonly accessed webserver for a scan. The tcpdump filter is simple:

(not src net 192.168) and not (tcp and src port 80)

This filter is then run with the -n and -t options set on tcpdump. The -n option tells tcpdump not to resolve the IP addresses into hostnames and the -t option suppresses the timestamp in the printed output. The result is piped to a file that is then sent to sort -u (to discard duplicates while sorting according to IP address).

We are able to sort by source IP address because we have thrown away the timestamp and hence the source IP is the first thing on the data line. Further, since we have ignored the timestamp, connections between hosts that are identical except for when they occurred will not be considered. In this way, we are able to exploit the efficient coding of the sort routine (which is much faster than perl) and reduce the data further by throwing away duplicate connections.

The final step is to run a perl program that examines the filtered sorted unique output to determine which source hosts contacted many destination hosts. Because we have sorted the data according to source IP address, we are able to sidestep the time-consuming process of searching for the source IP in the list we construct each time a new data line is considered.

The final challenge then is to provide for the case where many, many (thousands or more) of destination hosts are contacted by a single source IP. To handle this case we attempted to reduce the search space by using an embedded hash table structure. Under this scheme, for a given source IP, we search for the first octet of the destination IP in a hash that consists only of previously found first octet numbers (max number of elements is 256), then once the first octet is found, it points to another hash containing recorded second octet numbers (corresponding to the first octet). The hash of second octet numbers is searched (again max search space is 256 elements) and a match here points to a hash of third octet numbers and so on. The final value of the fourth octet hash is the number of times this destination host was accessed by this source. Again, the code is available at the NSWC web site.

Reducing the Data Volume

We have found there is a practical limit to the analysis that can be performed with filters. The tcpdump program collects a LOT of data. In order to perform more advanced analysis, it is necessary to reduce the data. We do this in two steps.

First, we use filters to separate the data by protocol. Then, we further reduce the data to a common format that we use to store information from any sensor.

The reduced data is comma delimited so it can easily be imported into databases or spreadsheets and contains the following fields: date, time, src, srcport, dst, dstport, protocol. We refer to this format as bcp, since it is Bulk CoPied from all analysis systems to a huge database for storage or historical analysis.

Our first step is to convert from hourly to daily files. We do this because, in comparing historical information, delays are created by the overhead involved in opening and closing hourly files. Here s how we do conversion and reduction.

We use tcpslice to concatenate hourly files into daily files. We can put a whole day s worth of files together using tcpslice. From our /LOG/date, the directory holding the day s data we type:

$ tcpslice *.9802* > tcpdump.980202

Tcpslice cats all the hourly files into a daily file tcpdump.980202, the daily file for February 02, 1998.

Next, we extract the various protocols we are interested in as part of the data reduction. At our site the primary protocols in use are: TCP, UDP, ICMP, IGMP and IGRP. To extract UDP into an ascii file for further processing:

$ tcpdump -r tcpdump.980202 udp > udp.980202

We repeat this operation for each protocol we are interested in keeping for long term analysis. We run some cursory tests for ICMP looking for routing updates that are from external addresses, but do not archive IGRP or IGMP. We do reduce and archive for the UDP, TCP, and ICMP protocols.

Then we run a simple test on the data.

$ tcpdump -r tcpdump.980202 "(not udp and not tcp and not icmp and not igrp and not igmp)" > other.980202

From time to time we find some interesting IP traffic that is not one of the protocols.

If other is not an empty file then it would pay to invest some time tracking down the source and destination addresses to sort out what is going on. There are a variety of services including hardware encryption units that use IP datagrams of various types in order to communicate.

While it is important to know how to do all these things, it does get monotonous, so we have included the consolidate.pl script to take all the hourly files and produce daily files from them.

Finally, in order to compare results over time, we convert the protocol files to bcp format using translate.pl. Another advantage of converting files to this format is that its universal format allows us to integrate data from different kinds of sensors.

SECTION 6 Pattern Analysis For Intrusion Detection *(By Example)*

Pattern matching is a powerful tool for network intrusion detection. Sensor systems placed at strategic locations can provide a wealth of information full of potentially important patterns to the analyst who seeks to protect and efficiently operate a network connected to the Internet.

This section focuses on intrusion detection techniques using data collected from a Linux-based sensor running on a Pentium PC positioned outside the firewall of a large DOD organization. Data is collected via tcpdump, a publicly available software package based on the libpcap library. We consider protocols from the TCP/IP protocol suite, and look both for attacks having known signatures and for anomalous network traffic by defining what normal traffic looks like.

Six types of attacks and detection methods are discussed:

- ¥ Denial of Service Attacks
- ¥ Network Vulnerability Scanning
- ¥ Machine Vulnerability Scanning
- ¥ Network Mapping / Information Collection
- ¥ Filtering for intrusion detection with tcpdump
- ¥ Subtle and Stealthy Attacks: detection and interpretation

Denial of Service Attack Patterns

The simplest and most commonly executed denial of service attack causes a machine to crash or grind to a halt by barraging it with icmp echo requests or replies. In the example shown in Pattern 1, the IP address spoofed.pound.me.net is spoofed by the attacker. The Class C nets 192.168.15 and 192.168.1 receive broadcast echo requests and every machine on the network replies to the broadcast. By spacing the requests closely in time, the attacker causes spoofed.pound.me.net to crash under the barrage of echo replies. Meanwhile, the 192.168.1 and 192.168.15 nets become clogged with traffic and the machines slow their processing tasks in order to devote resources to responding to the echo requests. The common name for this denial of service is the Smurf attack, and the CERT advisory is number CA-98.01. The distinguishing characteristics of the Smurf attack are the spoofed source IP address and the direction of the echo requests to broadcast addresses.

The pattern of alternating the x.x.15 and x.x.1 subnets is commonly seen. It is not known why these two subnets are so often targeted, except that these subnets are often used for infrastructure machines.

In Pattern 1 below, the format is:
Timestamp SourceIP > DestinationIP: icmp

Pattern 1:

```
00:00:05.327 spoofed.pound.me.net > 192.168.15.255: icmp: echo request
00:00:05.342 spoofed.pound.me.net > 192.168.1.255:  icmp: echo request
00:00:14.154 spoofed.pound.me.net > 192.168.15.255: icmp: echo request
00:00:14.171 spoofed.pound.me.net > 192.168.1.255:  icmp: echo request
00:00:19.055 spoofed.pound.me.net > 192.168.15.255: icmp: echo request
00:00:19.073 spoofed.pound.me.net > 192.168.1.255:  icmp: echo request
00:00:23.873 spoofed.pound.me.net > 192.168.15.255: icmp: echo request
00:00:23.889 spoofed.pound.me.net > 192.168.1.255:  icmp: echo request
00:00:33.503 spoofed.pound.me.net > 192.168.15.255: icmp: echo request
00:00:33.576 spoofed.pound.me.net > 192.168.1.255:  icmp: echo request
```

Pattern 2 shows another common icmp echo request attack pattern. In this case, the packets addressed to 255.255.255.255 are source-routed to our network and the 0.0 broadcasts target the older-style machines.

Pattern 3 shows another common sequence consisting only of the source-routed packets. These attacks generally continue at the rate suggested in the examples for hours and days at a time. Keep in mind that other networks may be included in these patterns besides your own if there is any lag at all between requests.

Pattern 2:

```
05:20:48.261 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:20:48.263 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:21:35.792 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:21:35.819 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:22:16.909 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:22:16.927 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:22:58.046 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:22:58.061 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:23:39.205 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:23:39.207 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:24:20.336 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:24:20.367 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:25:01.481 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:25:01.498 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:25:42.645 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
```

```
05:25:42.660 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:26:23.757 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:26:23.777 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:27:04.925 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:27:04.927 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:27:46.047 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:27:46.079 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
05:28:27.191 spoofed.pound.me.net > 192.168.0.0:     icmp: echo request
05:28:27.217 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
```

Pattern 3:

```
19:53:16.134 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:53:27.234 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:53:38.336 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:53:49.438 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:54:00.518 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:54:11.619 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:54:22.720 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
19:54:33.825 spoofed.pound.me.net > 255.255.255.255: icmp: echo request
```

The next example shows a half-successful icmp denial-of-service attack in action. In this case, Pattern 4, we see a router protecting the real host spoofed.pound.me.net from the barrage of echo replies being generated at our network. As the router discards the packets, it sends icmp error messages back to our 192.168.1.x machines. Note that the 192.168.1 net will be slowed by the burden of the echo traffic.

Pattern 4:

```
03:30:31.652433 router.pound.me.net > 192.168.1.1: icmp: host spoofed.pound.me.net unreachable
03:30:31.653107 router.pound.me.net > 192.168.1.2: icmp: host spoofed.pound.me.net unreachable
03:30:31.862459 router.pound.me.net > 192.168.1.1: icmp: host spoofed.pound.me.net unreachable
03:30:31.864012 router.pound.me.net > 192.168.1.3: icmp: host spoofed.pound.me.net unreachable
03:30:31.865914 router.pound.me.net > 192.168.1.4: icmp: host spoofed.pound.me.net unreachable
03:30:31.866646 router.pound.me.net > 192.168.1.5: icmp: host spoofed.pound.me.net unreachable
03:30:32.422559 router.pound.me.net > 192.168.1.1: icmp: host spoofed.pound.me.net unreachable
03:30:35.802449 router.pound.me.net > 192.168.1.6: icmp: host spoofed.pound.me.net unreachable
03:30:35.954542 router.pound.me.net > 192.168.1.6: icmp: host spoofed.pound.me.net unreachable
03:30:35.976808 router.pound.me.net > 192.168.1.5: icmp: host spoofed.pound.me.net unreachable
03:30:36.112835 router.pound.me.net > 192.168.1.2: icmp: host spoofed.pound.me.net unreachable
```

Pattern 5 shows the denial of service attack from yet another perspective. In this case, a router is refusing to route the broadcast echo requests before they reach the target networks. That is, the router is intercepting the packets on the way out. Assuming the packets originate somewhere near the far.away.com network, the router router.far.away.com stops the broadcast packets from proceeding and sends an icmp error message back to the source. However, since the source IP addresses are spoofed the error messages are routed to our networks, where the packets appear to have originated.

Pattern 5:

```
00:00:06.754739 router.far.away.com > 192.168.24.163: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:12.050836 router.far.away.com > 192.168.18.34: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:16.778674 router.far.away.com > 192.168.24.205: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:17.436264 router.far.away.com > 192.168.24.100: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:18.175952 router.far.away.com > 192.168.24.100: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:18.763302 router.far.away.com > 192.168.24.6: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:19.436072 router.far.away.com > 192.168.24.172: icmp: host 192.168.31.255 unreachable - admin prohibited filter
00:00:19.965784 router.far.away.com > 192.168.22.75: icmp: host 192.168.31.255 unreachable - admin prohibited filter
```

A somewhat more aggressive attack of this form is known as the ping-of-death attack. In this attack, the icmp echo requests (pings) are larger than they normally would be and hence must be fragmented. Many machines cannot handle fragmented or oversized icmp packets and will crash or freeze if they receive them. The CERT advisory for this attack is CA-96.26.

Pattern 6 shows a ping-of-death attack. In this particular example, big.pinger.org is targeting a web server on our network. The hostile source simultaneously attempts to crash the server with oversized ping packets and browses the web pages to see if the server is still alive. (The sequence numbers and other TCP specific information has been omitted for this example, only TCP active open connections are shown. Connections with destination port equal to 80 are http transactions.)

The active open (SYN) TCP connections are formatted according to:
Timestamp SourceIP.SourcePort > DestinationIP.DestinationPort: S

Pattern 6:

```
00:12:06.698 big.pinger.org.1369 > www.mynetwork.net.80: S
00:12:09.954 big.pinger.org > www.mynetwork.net: icmp: echo request (frag 60948:552@0+)
00:12:09.964 big.pinger.org > www.mynetwork.net:   (frag 60948:156@552)
00:13:39.452 big.pinger.org > www.mynetwork.net: icmp: echo request (frag 1813:552@0+)
00:13:39.472 big.pinger.org > www.mynetwork.net:   (frag 1813:156@552)
00:13:39.635 big.pinger.org.1371 > www.mynetwork.net.80: S
00:13:48.171 big.pinger.org.1373 > www.mynetwork.net.80: S
00:14:11.835 big.pinger.org > www.mynetwork.net: icmp: echo request (frag 11285:552@0+)
00:14:11.901 big.pinger.org > www.mynetwork.net:   (frag 11285:156@552)
00:14:11.950 big.pinger.org.1374 > www.mynetwork.net.80: S
```

The last denial of service attack pattern that we will discuss is the echo-chargen scan. This attack is illustrated in pattern 7. Here, the source IP address is again spoofed and the idea is to deny service to both the source and target. The source sends a request to the target s character generator port (19) using echo (7) as a source port. The target responds to the query by sending a stream of random characters back to the echo port of the source machine. The source then echoes the random characters back to the character generator that in turn causes another random character string to be generated. This pattern continues until one of the machines terminates the loop. The apparent source machine will have difficulty responding to all of the echo requests, and the target networks will be clogged with the traffic as long as spoofed.pound.me.net responds.

In Pattern 7 below the format is:
Timestamp SourceIP.SourcePort > DestinationIP.DestinationPort: udp

Pattern 7:

```
08:08:16.155354 spoofed.pound.me.net.echo > 172.31.203.17.chargen: udp
08:21:48.891451 spoofed.pound.me.net.echo > 192.168.14.50.chargen: udp
08:25:12.968929 spoofed.pound.me.net.echo > 192.168.102.3.chargen: udp
08:25:24.779435 spoofed.pound.me.net.echo > 192.168.102.3.chargen: udp
08:42:22.605428 spoofed.pound.me.net.echo > 192.168.18.28.chargen: udp
08:42:33.934939 spoofed.pound.me.net.echo > 192.168.18.28.chargen: udp
08:47:21.450708 spoofed.pound.me.net.echo > 172.31.130.93.chargen: udp
08:47:32.685521 spoofed.pound.me.net.echo > 172.31.130.93.chargen: udp
08:51:27.491458 spoofed.pound.me.net.echo > 172.31.153.78.chargen: udp
08:53:13.530992 spoofed.pound.me.net.echo > 172.31.46.49.chargen: udp
08:53:25.282020 spoofed.pound.me.net.echo > 172.31.46.49.chargen: udp
09:01:28.285296 spoofed.pound.me.net.echo > 172.31.149.67.chargen: udp
09:01:39.889736 spoofed.pound.me.net.echo > 172.31.149.67.chargen: udp
09:05:46.196868 spoofed.pound.me.net.echo > 192.168.154.96.chargen: udp
09:05:58.240337 spoofed.pound.me.net.echo > 192.168.154.96.chargen: udp
```

Network Vulnerability Scanning Attack Patterns

Often we will observe a source host scanning a network for machines running a certain type of service. Pattern 8 shows a hostile source scanning our 192.168.1 subnet for machines running imap (port 143), netbios (port 139), Socks (port 1080), and DNS services (port 53). These are all TCP connections and only the initial active open (SYN) connection has been shown. Notice the pattern of incrementing destination IP addresses and the short time interval between connection attempts.

Pattern 8:

```
10:53:57.951372 hostile.scan.com.7301 > 192.168.1.46.143: S
10:53:57.993072 hostile.scan.com.7367 > 192.168.1.47.143: S
10:53:57.997391 hostile.scan.com.7418 > 192.168.1.48.143: S
10:53:57.998690 hostile.scan.com.7421 > 192.168.1.54.143: S
10:53:58.001198 hostile.scan.com.7430 > 192.168.1.53.143: S
10:53:58.002513 hostile.scan.com.7496 > 192.168.1.52.143: S
10:53:59.947582 hostile.scan.com.10101 > 192.168.1.46.139: S
10:53:59.986003 hostile.scan.com.10167 > 192.168.1.47.139: S
10:53:59.993731 hostile.scan.com.10233 > 192.168.1.49.139: S
10:53:59.997864 hostile.scan.com.10299 > 192.168.1.50.139: S
10:53:59.999141 hostile.scan.com.10344 > 192.168.1.51.139: S
10:54:00.000779 hostile.scan.com.10410 > 192.168.1.52.139: S
10:54:01.941514 hostile.scan.com.12645 > 192.168.1.46.1080: S
10:54:01.985726 hostile.scan.com.12654 > 192.168.1.47.1080: S
10:54:01.993694 hostile.scan.com.12672 > 192.168.1.48.1080: S
10:54:01.996534 hostile.scan.com.12709 > 192.168.1.54.1080: S
10:54:01.997801 hostile.scan.com.12718 > 192.168.1.53.1080: S
10:54:01.999738 hostile.scan.com.12736 > 192.168.1.52.1080: S
10:54:03.976128 hostile.scan.com.15440 > 192.168.1.46.53: S
10:54:04.014408 hostile.scan.com.15486 > 192.168.1.47.53: S
10:54:04.248110 hostile.scan.com.15499 > 192.168.1.49.53: S
10:54:04.249351 hostile.scan.com.15554 > 192.168.1.50.53: S
10:54:04.250628 hostile.scan.com.15620 > 192.168.1.51.53: S
10:54:04.251952 hostile.scan.com.15686 > 192.168.1.52.53: S
```

Often we can identify a scan as resulting from the use of a certain (well-circulated) script. Patterns 9 and 10 show common scripted exploits widely circulated on the Internet. Pattern 9 shows a scan for imap servers, the script signature is the source port of 10143. Pattern 10 shows another scripted imap network scan where the source port of 0 is used and the SYN and FIN flags are both set on the TCP packets. The purpose of the unusual TCP flag combination is presumably to avoid being logged by sensors set to look for SYN-only packets (e.g., TAMU s Netlogger). Also notice that the packet sequence numbers is the same for each host contacted. In both patterns, the attacker is looking for responses from the target machines that indicate that an imap server is running.

Pattern 9:

```
14:13:54.847401 newbie.hacker.org.10143 > 192.168.1.1.143: S
14:24:58.151128 newbie.hacker.org.10143 > 172.31.1.1.143: S
14:35:40.311513 newbie.hacker.org.10143 > 192.168.1.2.143: S
14:43:55.459380 newbie.hacker.org.10143 > 192.168.2.1.143: S
14:54:58.693768 newbie.hacker.org.10143 > 172.31.2.1.143: S
15:05:41.039905 newbie.hacker.org.10143 > 192.168.2.2.143: S
15:13:59.948065 newbie.hacker.org.10143 > 192.168.3.1.143: S
15:25:03.215362 newbie.hacker.org.10143 > 172.31.3.1.143: S
15:35:45.336385 newbie.hacker.org.10143 > 192.168.3.2.143: S
15:54:59.309457 newbie.hacker.org.10143 > 172.31.4.1.143: S
```

Pattern 10:

```
13:10:33.281198 newbie.hacker.org.0 > 192.168.26.203.143: SF 374079488:374079488(0) win 512
13:10:33.334983 newbie.hacker.org.0 > 192.168.24.209.143: SF 374079488:374079488(0) win 512
13:10:33.357565 newbie.hacker.org.0 > 192.168.17.197.143: SF 374079488:374079488(0) win 512
13:10:33.378115 newbie.hacker.org.0 > 192.168.16.181.143: SF 374079488:374079488(0) win 512
```

More Machine Vulnerability Scanning Attack Patterns

In addition to scanning a network for machines running a particular type of server, we can consider a single machine and attempt to discern what services are running on it. Pattern 11 shows an excerpt from a portscan where all TCP ports were queried for services on a single target machine. The objective is to determine which ports have services running on them. Once an attacker knows what services are available to work with, he can further focus the attack.

Pattern 11:

```
09:52:20.811462 bad.guy.org.1788 > target.mynetwork.com.18: S
09:52:20.844837 bad.guy.org.1789 > target.mynetwork.com.17: S
09:52:20.876586 bad.guy.org.1790 > target.mynetwork.com.16: S
09:52:20.903545 bad.guy.org.1791 > target.mynetwork.com.15: S
09:52:24.367002 bad.guy.org.1792 > target.mynetwork.com.14: S
09:52:24.395297 bad.guy.org.1796 > target.mynetwork.com.13: S
09:52:25.349706 bad.guy.org.1797 > target.mynetwork.com.12: S
09:52:25.375756 bad.guy.org.1798 > target.mynetwork.com.11: S
09:52:26.573678 bad.guy.org.1800 > target.mynetwork.com.10: S
```

09:52:26.603163 bad.guy.org.1802 > target.mynetwork.com.9: S
09:52:28.639922 bad.guy.org.1804 > target.mynetwork.com.8: S
09:52:28.668172 bad.guy.org.1806 > target.mynetwork.com.7: S
09:52:32.749958 bad.guy.org.1808 > target.mynetwork.com.6: S
09:52:32.772739 bad.guy.org.1809 > target.mynetwork.com.5: S
09:52:32.802331 bad.guy.org.1810 > target.mynetwork.com.4: S
09:52:32.824582 bad.guy.org.1812 > target.mynetwork.com.3: S
09:52:32.850126 bad.guy.org.1814 > target.mynetwork.com.2: S
09:52:32.871856 bad.guy.org.1816 > target.mynetwork.com.1: S
09:52:32.923555 bad.guy.org.1819 > target.mynetwork.com.79: S
09:52:34.203913 bad.guy.org.1821 > target.mynetwork.com.78: S
09:52:36.212493 bad.guy.org.1822 > target.mynetwork.com.77: S
09:52:36.236889 bad.guy.org.1823 > target.mynetwork.com.76: S
09:52:36.264028 bad.guy.org.1824 > target.mynetwork.com.75: S
09:52:36.286556 bad.guy.org.1825 > target.mynetwork.com.74: S
09:52:36.314418 bad.guy.org.1826 > target.mynetwork.com.73: S
09:52:36.337039 bad.guy.org.1827 > target.mynetwork.com.72: S
09:52:36.366641 bad.guy.org.1828 > target.mynetwork.com.71: S
09:52:36.390532 bad.guy.org.1830 > target.mynetwork.com.70: S
09:52:36.413112 bad.guy.org.1831 > target.mynetwork.com.69: S
09:52:36.435624 bad.guy.org.1832 > target.mynetwork.com.68: S
09:52:36.457088 bad.guy.org.1833 > target.mynetwork.com.67: S
09:52:36.481007 bad.guy.org.1834 > target.mynetwork.com.66: S
09:52:36.508669 bad.guy.org.1835 > target.mynetwork.com.65: S
09:52:36.531770 bad.guy.org.1836 > target.mynetwork.com.64: S
09:52:36.555682 bad.guy.org.1837 > target.mynetwork.com.63: S
09:52:36.581866 bad.guy.org.1838 > target.mynetwork.com.62: S

The hacker community has developed and distributed special scanning tools that look for some of the well-known (exploitable) services. The portscan resulting from these tools is usually limited to specific well-known ports. For example, the portscan in Pattern 12 probes for only 6 services: telnet (port 23), http (port 80), imap (port 143), dns (port 53), pop-3 (port 110), and sunrpc (port 111).

Pattern 12:

```
06:13:23.188197 bad.guy.org.6479  > target.mynetwork.com.23:  S
06:13:26.150250 bad.guy.org.6479  > target.mynetwork.com.23:  S
06:13:28.071161 bad.guy.org.15799 > target.mynetwork.com.80:  S
06:13:31.075201 bad.guy.org.15799 > target.mynetwork.com.80:  S
06:13:33.107599 bad.guy.org.25467 > target.mynetwork.com.143: S
06:13:36.071352 bad.guy.org.25467 > target.mynetwork.com.143: S
06:13:38.068035 bad.guy.org.3861  > target.mynetwork.com.53:  S
06:13:41.219400 bad.guy.org.3861  > target.mynetwork.com.53:  S
06:13:43.271220 bad.guy.org.14296 > target.mynetwork.com.110: S
06:13:47.831695 bad.guy.org.943   > target.mynetwork.com.111: S
06:13:50.714731 bad.guy.org.943   > target.mynetwork.com.111: S
06:13:56.737729 bad.guy.org.943   > target.mynetwork.com.111: S
```

The two portscans shown in Patterns 11 and 12 involve TCP ports and services only. UDP portscans are rarer and usually more erratic. Pattern 13 shows an example of a UDP port scan.

In Pattern 13 below the format is:
Timestamp SourceIP SourcePort > DestinationIP DestinationPort :udp

Pattern 13:

```
18:20:29  bad.guy.org  22555  >  target.mynetwork.com  22555 :udp
18:20:35  bad.guy.org  10823  >  target.mynetwork.com  32927 :udp
18:20:36  bad.guy.org  45402  >  target.mynetwork.com  33866 :udp
18:20:37  bad.guy.org  46211  >  target.mynetwork.com   1318 :udp
18:20:37  bad.guy.org  12257  >  target.mynetwork.com  50019 :udp
18:20:37  bad.guy.org  41530  >  target.mynetwork.com  49246 :udp
18:20:38  bad.guy.org  31696  >  target.mynetwork.com  63627 :udp
18:20:39  bad.guy.org  55075  >  target.mynetwork.com  13893 :udp
18:20:39  bad.guy.org  17032  >  target.mynetwork.com   5292 :udp
18:20:40  bad.guy.org  30381  >  target.mynetwork.com  57795 :udp
18:20:40  bad.guy.org  44073  >  target.mynetwork.com  49844 :udp
18:20:40  bad.guy.org  26233  >  target.mynetwork.com  62829 :udp
18:20:40  bad.guy.org    332  >  target.mynetwork.com  49489 :udp
18:20:41  bad.guy.org  33057  >  target.mynetwork.com  50827 :udp
18:20:41  bad.guy.org  53888  >  target.mynetwork.com    520 :udp
18:20:41  bad.guy.org  52491  >  target.mynetwork.com  60539 :udp
```

```
18:20:41  bad.guy.org  20429  >  target.mynetwork.com  36655 :udp
18:20:42  bad.guy.org  28036  >  target.mynetwork.com  21409 :udp
18:20:42  bad.guy.org  10648  >  target.mynetwork.com  58237 :udp
18:20:42  bad.guy.org  33743  >  target.mynetwork.com  23258 :udp
18:20:42  bad.guy.org  43690  >  target.mynetwork.com  43690 :udp
18:20:42  bad.guy.org  35603  >  target.mynetwork.com  15080 :udp
18:20:43  bad.guy.org   2077  >  target.mynetwork.com  61741 :udp
18:20:43  bad.guy.org  15434  >  target.mynetwork.com   2131 :udp
18:20:43  bad.guy.org  31954  >  target.mynetwork.com  53798 :udp
18:20:43  bad.guy.org  39684  >  target.mynetwork.com      0 :udp
18:20:43  bad.guy.org  20364  >  target.mynetwork.com  33680 :udp
18:20:43  bad.guy.org  10638  >  target.mynetwork.com  12922 :udp
18:20:43  bad.guy.org   1344  >  target.mynetwork.com   8068 :udp
18:20:44  bad.guy.org  33044  >  target.mynetwork.com  19465 :udp
18:20:44  bad.guy.org  21534  >  target.mynetwork.com  60981 :udp
```

Network Mapping Attack Patterns

Mapping is an attempt, on the part of a hostile party, to gather information about a network; specifically, what IP addresses are alive and what types of machines (in terms of hardware, operating system, etc.) are running at what addresses. The simplest type of mapping is based on icmp echo-requests. The interested party simply attempts to ping each machine in the network, either one machine at a time or through reasonably spaced, broadcasted echo requests. This tells the source what machines are up and running on the network. Patterns 14 and 15 show excerpted examples of icmp-based network mapping. In the case of Pattern 14, the entire Class B address space was pinged, one machine at a time at the average rate of 250 machines per hour.

Pattern 14:

```
00:58:48.330217 pinger.mappem.com > 192.168.5.247: icmp: echo request
00:59:00.409576 pinger.mappem.com > 192.168.5.248: icmp: echo request
00:59:13.673834 pinger.mappem.com > 192.168.5.249: icmp: echo request
00:59:26.860533 pinger.mappem.com > 192.168.5.250: icmp: echo request
00:59:39.991059 pinger.mappem.com > 192.168.5.251: icmp: echo request
00:59:53.281628 pinger.mappem.com > 192.168.5.252: icmp: echo request
01:00:06.390525 pinger.mappem.com > 192.168.5.253: icmp: echo request
01:00:19.463582 pinger.mappem.com > 192.168.5.254: icmp: echo request
01:00:32.563453 pinger.mappem.com > 192.168.5.255: icmp: echo request
01:00:35.719236 pinger.mappem.com > 192.168.6.0: icmp: echo request
01:00:38.861865 pinger.mappem.com > 192.168.6.1: icmp: echo request
```

01:00:51.903375 pinger.mappem.com > 192.168.6.2: icmp: echo request
01:01:04.925395 pinger.mappem.com > 192.168.6.3: icmp: echo request
01:01:18.014343 pinger.mappem.com > 192.168.6.4: icmp: echo request
01:01:31.035095 pinger.mappem.com > 192.168.6.5: icmp: echo request
01:01:44.078728 pinger.mappem.com > 192.168.6.6: icmp: echo request
01:01:57.098411 pinger.mappem.com > 192.168.6.7: icmp: echo request
01:02:10.141218 pinger.mappem.com > 192.168.6.8: icmp: echo request
01:02:23.170212 pinger.mappem.com > 192.168.6.9: icmp: echo request
01:02:36.218773 pinger.mappem.com > 192.168.6.10: icmp: echo request
01:02:49.262484 pinger.mappem.com > 192.168.6.11: icmp: echo request
01:03:02.311444 pinger.mappem.com > 192.168.6.12: icmp: echo request

Pattern 15:

00:43:58.094644 pinger.mappem.com > 192.168.64.255: icmp: echo request
00:43:58.604889 pinger.mappem.com > 192.168.64.0: icmp: echo request
00:43:59.060335 pinger.mappem.com > 192.168.64.255: icmp: echo request
00:43:59.578614 pinger.mappem.com > 192.168.64.0: icmp: echo request
00:50:02.297035 pinger.mappem.com > 192.168.65.255: icmp: echo request
00:50:02.689911 pinger.mappem.com > 192.168.65.0: icmp: echo request
00:50:03.278235 pinger.mappem.com > 192.168.65.255: icmp: echo request
00:50:03.685129 pinger.mappem.com > 192.168.65.0: icmp: echo request
00:54:56.911891 pinger.mappem.com > 192.168.66.255: icmp: echo request
00:54:57.265833 pinger.mappem.com > 192.168.66.0: icmp: echo request
00:54:57.909921 pinger.mappem.com > 192.168.66.255: icmp: echo request
00:54:58.250772 pinger.mappem.com > 192.168.66.0: icmp: echo request
00:59:52.822243 pinger.mappem.com > 192.168.67.255: icmp: echo request
00:59:53.415182 pinger.mappem.com > 192.168.67.0: icmp: echo request
00:59:53.790773 pinger.mappem.com > 192.168.67.255: icmp: echo request
00:59:54.410082 pinger.mappem.com > 192.168.67.0: icmp: echo request

Along this line is the obvious extension to TCP or UDP echo mapping. Pattern 16 shows an example of mapping using the UDP echo request. Notice that this mapping example is of the low-and-slow variety, where only a few requests are received each hour, rather than many per second. The ICMP echo mapping attempts are observed much more frequently than UDP or TCP echo mapping endeavors.

Pattern 16:

```
01:02:06.614308 slowpoke.mappem.com.2176 > 172.31.66.84.echo: udp 6
01:04:03.948045 slowpoke.mappem.com.2176 > 192.168.34.226.echo: udp 6
01:15:24.853282 slowpoke.mappem.com.2176 > 192.168.6.103.echo: udp 6
01:31:03.166110 slowpoke.mappem.com.2176 > 172.31.187.209.echo: udp 6
01:33:42.984626 slowpoke.mappem.com.2176 > 192.168.205.84.echo: udp 6
01:40:07.945375 slowpoke.mappem.com.3066 > 172.31.251.88.echo: udp 6
01:51:50.709035 slowpoke.mappem.com.3066 > 172.31.88.12.echo: udp 6
01:53:06.490632 slowpoke.mappem.com.3066 > 172.31.59.224.echo: udp 6
01:55:09.359423 slowpoke.mappem.com.3066 > 172.31.137.22.echo: udp 6
01:59:13.287870 slowpoke.mappem.com.3066 > 192.168.108.255.echo: udp 6
02:08:48.088681 slowpoke.mappem.com.3066 > 192.168.134.117.echo: udp 6
02:15:04.539055 slowpoke.mappem.com.3066 > 172.31.73.1.echo: udp 6
02:15:13.155988 slowpoke.mappem.com.3066 > 172.31.16.152.echo: udp 6
02:22:38.573703 slowpoke.mappem.com.3066 > 192.168.91.18.echo: udp 6
02:27:07.867063 slowpoke.mappem.com.3066 > 172.31.2.176.echo: udp 6
02:30:38.220795 slowpoke.mappem.com.3066 > 192.168.5.103.echo: udp 6
02:49:31.024008 slowpoke.mappem.com.3066 > 172.31.152.254.echo: udp 6
02:49:55.547694 slowpoke.mappem.com.3066 > 192.168.219.32.echo: udp 6
```

The mapping patterns we have discussed so far are all internal mapping attempts. By this it is meant that a hostile party is attempting to learn information about the machines inside the protected network. However, the wary administrator will want to know when their network is the target of external mapping attempts as well. External mapping means that a hostile party is attempting to learn information about the machines outside (but near) the protected network.

The information specifically sought in external mapping is exactly how many gateways connect the protected network to the Internet, along with as much information about these gateways as possible. In theory, if all gateways into and out of a protected network were incapacitated, the protected network would be completely isolated from the Internet. Thus, the protected network would be subject to a denial of service, regardless of the level of security employed. This can be particularly harmful if corporations or operational DOD facilities become completely disconnected, unable to even send or receive electronic mail.

The fundamental service on which external mapping is based is the traceroute function. Pattern 17 shows an example of external mapping using traceroute as logged by tcpdump. See W. Richard Stevens, *TCP/IP Illustrated Vol. 1*, Chapter 8 for a good description of the traceroute service and the associated tcpdump data trace. Basically, traceroute uses a combination of UDP datagrams and ICMP error messages to provide a listing of the routers that handle the UDP packet on the way from source to destination.

In pattern 17 we see four separate traceroute patterns. Each of four mappem.com machines (named north, south, east, and west) utilize the traceroute function to list the network hops on the route between themselves and the nameserver ns.target.net. In practice, the four mappem.com machines would be positioned on different class A nets, ideally in different parts of the world, relying on different pieces of the Internet architecture.

Pattern 17:

```
10:32:24.722 north.mappem.com.38758 > ns.target.net.33476: udp 12
10:32:24.756 north.mappem.com.38758 > ns.target.net.33477: udp 12
10:32:24.801 north.mappem.com.38758 > ns.target.net.33478: udp 12
10:32:24.833 north.mappem.com.38758 > ns.target.net.33479: udp 12
10:32:24.944 north.mappem.com.38758 > ns.target.net.33481: udp 12
10:32:24.975 north.mappem.com.38758 > ns.target.net.33482: udp 12
10:32:25.078 north.mappem.com.38758 > ns.target.net.33484: udp 12
10:32:26.541 south.mappem.com.48412 > ns.target.net.33510: udp 12
10:32:26.745 south.mappem.com.48412 > ns.target.net.33512: udp 12
10:32:26.837 south.mappem.com.48412 > ns.target.net.33513: udp 12
10:32:26.930 south.mappem.com.48412 > ns.target.net.33514: udp 12
10:32:27.033 south.mappem.com.48412 > ns.target.net.33515: udp 12
10:32:27.231 south.mappem.com.48412 > ns.target.net.33517: udp 12
10:32:27.436 south.mappem.com.48412 > ns.target.net.33519: udp 12
10:32:26.425 east.mappem.com.58853 > ns.target.net.33490: udp 12
10:32:26.541 east.mappem.com.58853 > ns.target.net.33491: udp 12
10:32:26.744 east.mappem.com.58853 > ns.target.net.33493: udp 12
10:32:26.836 east.mappem.com.58853 > ns.target.net.33494: udp 12
10:32:26.930 east.mappem.com.58853 > ns.target.net.33495: udp 12
10:32:27.033 east.mappem.com.58853 > ns.target.net.33496: udp 12
10:32:27.232 east.mappem.com.58853 > ns.target.net.33498: udp 12
10:32:27.323 east.mappem.com.58853 > ns.target.net.33499: udp 12
10:32:45.760 west.mappem.com.34081 > ns.target.net.33485: udp 12
10:32:45.958 west.mappem.com.34081 > ns.target.net.33486: udp 12
10:32:46.169 west.mappem.com.34081 > ns.target.net.33487: udp 12
10:32:46.425 west.mappem.com.34081 > ns.target.net.33488: udp 12
10:32:46.638 west.mappem.com.34081 > ns.target.net.33489: udp 12
10:32:46.850 west.mappem.com.34081 > ns.target.net.33490: udp 12
10:32:47.080 west.mappem.com.34081 > ns.target.net.33491: udp 12
10:32:47.271 west.mappem.com.34081 > ns.target.net.33492: udp 12
10:32:47.498 west.mappem.com.34081 > ns.target.net.33493: udp 12
10:32:47.720 west.mappem.com.34081 > ns.target.net.33494: udp 12
```

As shown, the traceroutes are logged within seconds of each other. Presumably, the traceroute commands were issued simultaneously, and it is the difference in datagram routing that results in the different logging times. Because the mappem.com machines are far apart (in terms of position on the Internet), the traceroutes from these machines will likely result in very different route listings. However, it might be imagined, that if one were to execute such traceroutes over and over, periodically, from several different vantage points one might gain a fair amount of insight into the routing configuration external to the network of interest.

In practice, this repetitious pattern is exactly what we observe. Several hostile machines simultaneously issue traceroute commands to an infrastructure machine on the protected network. These simultaneous traceroutes are executed every few days, at all different times of the day and night, and on weekends as well as workdays. The simultaneous execution helps to control the process, that is, if something looks unusual on one of the traces, the other traces may be considered to gain insight into the cause of the anomaly. In this manner, a resourceful individual may collect a great deal of information about how Internet Service Providers sustain a network s connection to the Internet.

Filtering for Intrusion Detection using tcpdump

The freely available software tcpdump provides a powerful filtering capability in addition to its data collection service. A high-level logic-based language allows the analyst to build powerful filters capable of detecting attacks, probes, and other events of interest quickly in large tcpdump log files. In fact, each of the attack patterns we have discussed so far can be detected with fairly simple tcpdump filters.

Because tcpdump has a limit on the maximum size a filter may be (based on the number of available registers) we break the filtering task up into five pieces. Four of these pieces are designated by protocol: there are filters for IP, icmp, TCP, and udp level signatures. The fifth piece corresponds to a filter that characterizes acceptable traffic flow to and from a few important machines on the network. The filter is instructed to report any traffic involving these machines that does not fit the acceptable traffic profile. Good machines to monitor are infrastructure machines, e.g., name servers, mail servers, http and ftp servers, access gateways, firewalls, and sensors. Because many attacks target such infrastructure machines, this fifth filter often uncovers new intrusion signatures.

Template versions of these filters are part of the SHADOW source distribution. A listing of some of the components that may be logically combined to create composite filters is given below.

IP Filters (Again)

We have examined several example filters; let s consider how we apply them in analysis. At this level, we may check to see if the source and destination IP addresses are the same. If they are, this is the signature of a land attack. Also, we may look for packets destined to an entire network via a broadcast destination address. Often icmp mapping and denial-of-service (Smurf) attacks are detected in this manner.

It is a good idea to monitor for fragmented IP datagrams. In particular, we want to be looking for fragmented icmp traffic, as it is very unusual and is usually hostile when observed. Further, the teardrop attack exploits a vulnerability in the way TCP/IP reassembles overlapping IP fragments. CERT advisory CA-97.28 reports on this exploit which enables a remote user to cause a denial-of-service. Finally, other fragmented traffic may also be of interest; for example, this filter often exposes the existence of an encrypted channel using an unusual IP protocol.

It can also be useful to monitor any activity logged involving unroutable IP addresses, for example net 127.x.x.x that is assigned to localhost. There are a few exploits that aim at gaining access to a machine by spoofing the localhost address. On the other hand, net 0.x.x.x traffic is typically indicative of a misconfiguration of a hardware board or piece of software installed on some machine on the subnet. Traffic from other reserved network addresses is of similar interest. Other address spaces that might be included here would be the 10.x.x.x, 192.168.x.x, and 172.16.x.x-172.31.x.x networks. Monitoring activity from these addresses usually uncovers leaks from (supposedly) isolated internal networks.

The last IP filter listed checks to see whether any IP options are set. An example of an IP option that is of interest is the source route option. Recall that in patterns 2 and 3 we inferred that the packets broadcast to the world at 255.255.255.255 were source-routed to our networks. It is a good idea to disallow source-routed packets in general, and thus it is a good thing to look for when monitoring. The only way to tell if IP options are set is to check the size of the IP header. If the header size is bigger than the no-option size of 20 bytes, then options have been set. The maximum size of the IP packet header is 60 bytes, so the option field may take up a maximum of 40 bytes. By checking the state of the first option flag (once we know that options have been set) we can determine what option was requested when the packet was sent.

IP filter atomic elements

```
#land attacks: source IP = dest IP
ip[12:4] = ip[16:4]

#broadcasts to x.x.x.255
ip[19] = 0xff

#broadcasts to x.x.x.0
ip[19] = 0x00

#fragmented IP packets with more fragments coming
ip[6:1] & 0x20 != 0

#fragmented ip packets with zero offset (first in sequence)
ip[6:2] & 0x1fff = 0

#unroutable ip addresses: 0.x.x.x, 127.x.x.x, 1.x.x.x, 2.x.x.x, #5.x.x.x, 240.x.x.x through 255.x.x.x
net 0
net 127
net 1
net 2
net 5
ip[12] > 239

#ip packets with options set (header size is bigger than 5 32-bit #words)
(ip[0:1] & 0x0f) > 5
```

if want to check for specific options, check ip[20:1] (given
#that (ip[0:1] & 0x0f > 5) is satisfied):

option	ip[20:1]
record route	7
timestamp	0x44
loose source routing	0x83
strict source routing	0x89

TCP filters

The TCP filters rely largely on the concept of ports. Usually we want to look at inbound active open connections only (SYN flag is set, ACK flag is not set). As we will see later, there are interesting patterns embedded in the SYN-ACK and RESET connections, however these are difficult to filter for at this level. Our efforts here are concentrated on the services requested as designated by the destination port used in the active open connection.

The list of interesting ports given here was compiled in part using the Appendix from Cheswick and Bellovin s book Firewalls and Internet Security. Other events of interest have been included in this list based on our personal experiences and data collection efforts. In all cases, the filters discussed here should be customized to apply specifically to the network they are utilized to monitor.

For example, although telnets are allowed to most networks in general, inbound telnet attempts to infrastructure machines might not ever be allowed. Under these circumstances, the filter should be modified to be something like:
(DST port 23) and
((DST host nameserver.mynet.net) or (DST host mailserver.mynet.net))

In that way, the filters will extract any telnet connection attempts to these special machines.

Further, in some cases it may be of interest to filter out telnet connections attempts from certain IP address spaces. In that case, the filter would be modified as given above, except constraints are then set in terms of src net.

Similarly, it is a good idea to screen for inbound nntp (Network News Transfer Protocol) connection attempts to machines that are not running nntp servers. Typically, isolated nntp probes are observed, rather than a noisy network scan; however, either way, the intent is to find machines running servers. At least one nntp server hack is in existence today.

DNS zone transfer (TCP port 53 connections) should be disallowed to all but trusted hosts. A TCP DNS connection indicates a request to download the current host table from the targeted nameserver. Such host table information can function as an excellent network map of active machines, and should thus be carefully guarded. We have observed both isolated connection attempts to known nameservers and also network-wide scans on TCP port 53. In the case of the isolated probes to known nameservers the attacker may be attempting to download the host table; however, in the case of the scans we postulate that the attacker is simply looking for DNS servers. Note that the vulnerabilities in many versions of BIND make DNS an attractive service to work with for many would-be hackers. CERT advisory CA-98.05 discusses three vulnerabilities in the BIND 4.9 and BIND 8 releases, whereby a hacker may gain root access or disrupt operations of a nameserver.

Well-known exploits also exist for IMAP and POP-3 (CA-98.09, CA-97.09) servers, implying that activity to these ports should be closely monitored if at all possible. In cases where legitimate servers of these protocols are running it is impossible to monitor every connection we will discuss an alternative method for those cases in the next section. However, in all cases it is possible to filter for the SYN-FIN, source port 0, TCP scans that are commonly used to scan for servers. See patterns 18 and 19 for examples of these scans, where the DNS and POP-3 ports are targeted, and compare to pattern 10 involving IMAP.

Pattern 18:

```
01:56:58.624019 script.junkie.net.0 > 192.168.93.0.53: SF   2216558592:2216558592(0) win 512
01:56:58.637759 script.junkie.net.0 > 192.168.93.1.53: SF 2216558592:2216558592(0) win 512
01:56:58.655110 script.junkie.net.0 > 192.168.93.2.53: SF 2216558592:2216558592(0) win 512
01:56:58.678034 script.junkie.net.0 > 192.168.93.3.53: SF 2216558592:2216558592(0) win 512
```

Pattern 19:

```
16:36:06.542659 script.junkie.net.0 > 192.168.26.203.110: SF 1681588224:1681588224(0) win 512
16:36:06.542966 script.junkie.net.0 > 192.168.18.84.110: SF 1681588224:1681588224(0) win 512
16:36:06.570371 script.junkie.net.0 > 192.168.43.254.110: SF 1681588224:1681588224(0) win 512
16:36:06.580533 script.junkie.net.0 > 192.168.24.209.110: SF 1681588224:1681588224(0) win 512
```

In terms of the NETBIOS ports, probes are most often observed to TCP port 139: Netbios Session Service. Scanning a network on TCP port 139 is a good way to locate SMB (Shared Message Block protocol, aka. Samba) file servers. Further information may be obtained from http://www.sabotage.org/rootshell in the paper CIFS: Common Insecurities Fail Scrutiny by Hobbit. This paper describes many protocol and administrative vulnerabilities in the NETBIOS file-sharing protocols. If your network includes Microsoft systems based on Windows NT, it is a good idea to check this out.

Of course it is smart to look for any inbound service requests, such as rlogin, rshell, rexec, and sunrpc (portmap). Further, a vulnerability in Secure Shell has been reported, so activity involving this port should be monitored. Additionally, we screen for window system ports and logins. By connecting to a window manager port (e.g., TCP port 6000), it is possible for a third party to observe the contents of the screen while the regular user types at the console. Also, it is useful to monitor activity involving TCP port 2049 (and udp port 2049); this port is used for NFS and isolated connections usually signify attempts to mount disks on the targeted system.

Hacker signature ports such as 31337, 87 and 95 are good alarm ports to filter for. Legitimate activity is typically not observed involving these ports. In hacker speak 31337 stands for the word ELEET (read elite), and this port is sometimes used as a source port in scripted attacks. Keying on these three ports will illuminate an occasional hostile connection, and will generate few false alarms. Pattern 20 shows a scan for http servers that can be distinguished from the noise because the signature source port of 31337 has been used.

Pattern 20:

```
18:54:22.737282 eleet.hacker.org.31337 > 192.168.220.1.80: S
18:54:22.738515 eleet.hacker.org.31337 > 192.168.220.3.80: S
18:54:22.740791 eleet.hacker.org.31337 > 192.168.220.7.80: S
18:54:22.742257 eleet.hacker.org.31337 > 192.168.220.5.80: S
18:54:22.743749 eleet.hacker.org.31337 > 192.168.220.9.80: S
18:54:22.745816 eleet.hacker.org.31337 > 192.168.220.11.80: S
18:54:22.750538 eleet.hacker.org.31337 > 192.168.220.13.80: S
18:54:22.755338 eleet.hacker.org.31337 > 192.168.220.15.80: S
18:54:22.758723 eleet.hacker.org.31337 > 192.168.220.17.80: S
18:54:22.761655 eleet.hacker.org.31337 > 192.168.220.19.80: S
18:54:22.763138 eleet.hacker.org.31337 > 192.168.220.21.80: S
```

The final destination-port-based filter included in this list is the look for anything unknown filter. This filter is an attempt to define the ports where we expect to see traffic, in order to illuminate any inbound active open connections to ports that we do not recognize. This is a good exercise to perform on large and unruly networks, since machines running servers on strange ports will quickly become apparent. This filter will also pick up connections involving software products that use a loosely defined set of TCP ports. For example, the MS NetMeeting software primarily uses TCP ports 1503 and 1720, but may use other ports as well when a NetMeeting is in session, an example of this is shown in pattern 21. Similarly, the ICQ chatting service uses several unassigned TCP ports that will trigger this filter. An example of the ICQ pattern is shown in pattern 22. A port list may be obtained from

ftp://ftp.isi.edu/pub/iana/port_numbers.txt

also the /etc/services file from the freeBSD distribution has an excellent port list.

In Patterns 21, 22, and 23 the format is:
Timestamp SourceIP SourcePort > DestinationIP DestinationPort: S

Pattern 21:

```
09:30:06 chatter.mynet.com 1084 > 127.25.233.30   1503: S
09:30:06 chatter.mynet.com 1084 > 127.25.233.30   1503: S
09:30:17 chatter.mynet.com 1085 > 127.25.233.30   1720: S
09:30:23 chatter.mynet.com 1086 > 127.25.233.30   1090: S
09:30:30 chatter.mynet.com 1088 > 127.25.233.30   1503: S
09:41:40 chatter.mynet.com 1136 > 172.26.172.171   1503: S
09:41:40 chatter.mynet.com 1136 > 172.26.172.171   1503: S
09:41:53 chatter.mynet.com 1137 > 172.26.172.171   1503: S
09:42:10 chatter.mynet.com 1138 > 172.26.172.171   1720: S
09:42:13 chatter.mynet.com 1138 > 172.26.172.171   1720: S
```

09:42:44 172.26.172.171 1695 > chatter.mynet.com 1720: S
09:42:48 172.26.172.171 1695 > chatter.mynet.com 1720: S
09:42:55 172.26.172.171 1696 > chatter.mynet.com 1140: S
09:49:52 10.149.1.44 1063 > chatter.mynet.com 1720: S
09:49:57 10.149.1.44 1064 > chatter.mynet.com 1141: S
09:50:12 10.149.1.44 1065 > chatter.mynet.com 1720: S
09:50:15 10.149.1.44 1065 > chatter.mynet.com 1720: S
09:50:30 10.149.1.44 1068 > chatter.mynet.com 1720: S
09:50:32 10.149.1.44 1069 > chatter.mynet.com 1143: S
09:57:01 10.149.1.44 1088 > chatter.mynet.com 1720: S
09:57:53 chatter.mynet.com 1146 > 251.31.171.115 1503: S
09:57:53 chatter.mynet.com 1146 > 251.31.171.115 1503: S
09:57:57 chatter.mynet.com 1147 > 251.31.171.115 1720: S
09:58:16 chatter.mynet.com 1148 > 251.31.171.115 2031: S
09:58:30 chatter.mynet.com 1150 > 251.31.171.115 1503: S
09:59:38 127.196.232.179 1259 > chatter.mynet.com 1503: S
09:59:44 127.196.232.179 1260 > chatter.mynet.com 1503: S

Pattern 22:

16:00:14 254.255.184.73 1086 > chatter.mynet.com 1028: S
16:00:14 10.99.54.159 1145 > chatter.mynet.com 1028: S
16:00:15 249.161.207.37 1500 > chatter.mynet.com 1028: S
16:00:17 10.99.54.159 1145 > chatter.mynet.com 1028: S
16:16:14 246.186.18.18 1804 > chatter.mynet.com 1028: S
16:16:16 10.121.20.190 1035 > chatter.mynet.com 1028: S
16:24:16 172.24.37.160 1364 > chatter.mynet.com 1028: S
16:25:31 chatter.mynet.com 1753 > 249.161.207.37 1479: S
16:25:34 chatter.mynet.com 1753 > 249.161.207.37 1479: S
16:25:41 chatter.mynet.com 1753 > 249.161.207.37 1479: S
16:25:53 chatter.mynet.com 1753 > 249.161.207.37 1479: S
16:36:18 247.53.89.112 1049 > chatter.mynet.com 1028: S
16:58:20 246.33.100.131 2650 > chatter.mynet.com 1028: S
16:58:26 10.76.206.180 1175 > chatter.mynet.com 1028: S
17:04:20 249.103.43.89 1065 > chatter.mynet.com 1028: S
17:04:20 248.140.58.105 1123 > chatter.mynet.com 1028: S
17:30:24 172.29.32.138 1044 > chatter.mynet.com 1028: S
17:31:44 chatter.mynet.com 2077 > 246.33.100.131 2451: S

```
17:31:45  chatter.mynet.com  2077 >     246.33.100.131  2451: S
17:31:45  chatter.mynet.com  2077 >     246.33.100.131  2451: S
17:31:46  chatter.mynet.com  2077 >     246.33.100.131  2451: S
17:32:30    247.214.199.216 1143 >  chatter.mynet.com  1028: S
17:40:25    172.19.142.13   1035 >  chatter.mynet.com  1028: S
17:40:25   10.237.129.136   1228 >  chatter.mynet.com  1028: S
18:08:27   254.65.238.206   1260 >  chatter.mynet.com  1028: S
18:15:45  chatter.mynet.com  2150 >      255.31.225.35  1035: S
18:16:02  chatter.mynet.com  2151 >    247.172.250.160  1027: S
18:16:26       10.24.3.162  1712 >  chatter.mynet.com  1028: S
18:16:38  chatter.mynet.com  2153 >     254.202.170.34  1245: S
18:16:41  chatter.mynet.com  2153 >     254.202.170.34  1245: S
18:16:52  chatter.mynet.com  2154 >      10.188.53.48   1099: S
18:18:18  chatter.mynet.com  2155 >    247.205.187.164  1027: S
18:18:45  chatter.mynet.com  2156 >      10.242.130.91  1067: S
18:18:59  chatter.mynet.com  2157 >      249.30.71.109  1894: S
18:19:09  chatter.mynet.com  2158 >      252.203.57.30  1281: S
18:19:20  chatter.mynet.com  2159 >    252.167.182.160  1306: S
18:19:28  chatter.mynet.com  2160 >      248.140.97.84  1029: S
18:19:31  chatter.mynet.com  2160 >      248.140.97.84  1029: S
18:20:29    172.24.144.32   1423 >  chatter.mynet.com  1028: S
18:22:35    10.65.180.31    1357 >  chatter.mynet.com  1028: S
18:22:38     10.65.180.31   1357 >  chatter.mynet.com  1028: S
18:30:14    172.24.144.32   1447 >  chatter.mynet.com  1028: S
```

Another application that may trigger the look for anything unknown filter falsely is FTP. As shown in pattern 23, a normal FTP data transfer initiated by a client at 192.168.16.3 results in the server performing several active opens to high numbered ports on the client machine. A security officer intent on protecting the 192.168.x.x network, might view these active open connections as hostile when first reported by the filter. However, when the full sequence of connections is considered, it is obvious that this is a FTP data download. In fact, this type of FTP data transfer is governed by the use of the PORT command. Under these circumstances, when the client connects to the server, it sends a PORT command to the server telling the server what port to open on the client machine. Meanwhile, the client performs a passive open on the high-numbered port. When a FTP data transfer is negotiated this way, the server almost always uses a source port of 20. Thus, it is easy to filter out these connections by ignoring traffic that trips the filter but has a source port equal to 20.

Pattern 23:

```
21:50:35    192.168.16.3  45472  > ftp.server.org    21:  S
21:51:33    ftp.server.org    20  >  192.168.16.3  45473:  S
21:52:14    ftp.server.org    20  >  192.168.16.3  45474:  S
21:52:31    ftp.server.org    20  >  192.168.16.3  45475:  S
21:53:18    ftp.server.org    20  >  192.168.16.3  45476:  S
21:53:38    ftp.server.org    20  >  192.168.16.3  45477:  S
21:54:15    ftp.server.org    20  >  192.168.16.3  45478:  S
21:54:57    ftp.server.org    20  >  192.168.16.3  45479:  S
21:55:13    ftp.server.org    20  >  192.168.16.3  45480:  S
21:55:38    ftp.server.org    20  >  192.168.16.3  45481:  S
21:56:57    ftp.server.org    20  >  192.168.16.3  45483:  S
21:57:16    ftp.server.org    20  >  192.168.16.3  45484:  S
```

Consider, however, the ftp data transfer shown in pattern 24. This example illustrates the case where the client machine resides behind a firewall (for example) and the server is prevented from performing active opens on the client machine. In order to get around this problem, the client sends a PASV command to the server when it connects. This command tells the server to choose a port for the data connection and send the port number back in the response. The server then performs a passive open on that port (on itself) and listens for a connection from the client. The client performs an active open to the server on the negotiated port, and thus the data connection is established. A description of this behavior is given in *TCP/IP: Architecture, Protocols, and Implementation* by Sidnie Feit. Note that we are unable to screen this traffic out based on the source port used in the connection, since the source port used will change for each data transfer.

However, when the problem is arranged this way, we see that the false alarms will result only when the filters detect active open TCP connections to high numbered ports on a FTP server that is resident on the protected network. Thus, if it is possible to identify the addresses of the FTP servers operating on the protected network (and capable of operating in PASV mode) it is a straightforward matter to customize the filters to ignore unusual TCP port connections to these servers.

Pattern 24:

```
06:11:21.798 client.shielded.com.1986 > ftp.server.mynet.net.21: S
06:11:25.840 client.shielded.com.1987 > ftp.server.mynet.net.47370: S
06:14:25.694 client.shielded.com.1988 > ftp.server.mynet.net.47377: S
06:17:10.857 client.shielded.com.1989 > ftp.server.mynet.net.47384: S
```

It is worthwhile to note that we have not specifically mentioned filtering for http (port 80), smtp (port 25), ftp (port 21), pop-3 (port 110) and other commonly used ports. This omission is certainly not due to the fact that there are no available exploits involving these services. Rather, the difficulty comes in distinguishing hostile TCP traffic to these ports from normal (or perhaps misconfigured) connections. At this level of filtering, it is virtually impossible to distinguish good from bad, and it is impossible to monitor all email or http connections, even on a modest-sized network. These issues will be addressed in the next section.

```
#active open packets (syn is set, ack is not)
(TCP[13] & 2 != 0) and (TCP[13] & 0x10 = 0)

#destination ports less than 20
TCP[2:2] < 20

#source ports less than 20
TCP[0:2] < 20

#ssh
dst port 22

#telnet (look for these from undesirable sources or to machines or
#          subnets that shouldn t be receiving telnets)
dst port 23

#whois
dst port 43

#dns zone transfer
dst port 53

#gopher
dst port 70

#finger
dst port 79

#link (hacker signature port)
(dst port 87) or (src port 87)

#supdup (hacker signature port)
(dst port 95) or (src port 95)

#sunrpc (portmapper)
dst port 111

#nntp (look for these to non-news servers)
dst port 119

#epmap   DCE Endpoint resolution
dst port 135

#profile   PROFILE naming system
dst port 136
```

```
#netbios name service
dst port 137

#netbios datagram service
dst port 138

#netbios session service
dst port 139

#imap
dst port 143

#NeWS window system
dst port 144

#exec
dst port 512

#login
dst port 513

#shell
dst port 514

#printer spooler (look for unusual activity)
dst port 515

#uucp
dst port 540

#doom and flash
dst port 666

#kerberos admin
dst port 749

#loadav
dst port 750

#pump
dst port 751

#Socks
dst port 1080

#openwin (like X11)
dst port 2000
```

```
#nfs
dst port 2049

#listen   the System V listener
dst port 2766

#Internet relay chat
dst port 6667

#X11 ports
(TCP[2:2] >= 6000) and (TCP[2:2] < 7000)

#eleet (hacker signature port)
(dst port 31337) or (src port 31337)

#syn and fin flags are set simultaneously
(TCP[13] & 2 != 0) and (TCP[13] & 1 !=0)

#urgent flag is set
TCP[13] & 0x20 !=0

#look for anything unknown
not ((TCP[2:2] < 20) or
        dst port 21 or
        dst port 22 or
        dst port 23 or
            dst port 25 or
            dst port 37 or
            dst port 43 or
            dst port 53 or
            dst port 70 or
            dst port 79
```

Subtle and Stealthy Attacks: detection and interpretation

In pattern 25, the remote host (stealth.mappem.com) sends a RESET packet to an internal address. The border router intercepts the packet and needs to make a decision. If the internal system exists, the router can forward the packet along. If the internal system doesn t exist, then the router replies with an unreachable message. Based on the responses from the border router, the attacker can map the internal active addresses. This is considered stealthy since the internal systems view RESET packets as normal traffic and will not log this activity. Unless an Intrusion Detection System is in place, this is truly a stealthy network mapping attempt.

Pattern 25:

```
03:00:58.939 stealth.mappem.com.35124 > 192.168.182.171.1626: R 0:0(0) ack 674719802 win 0
03:00:58.940 router.mynet.net > stealth.mappem.com: icmp: host 192.168.182.171 unreachable
```

A slightly different scenario is shown in pattern 26 where SYN-ACK packets are used instead of RESET connections. In this case, the hosts themselves (rather than the router) are tricked into giving up information. From the scan below, stealth.mappem.com will know that the hosts 192.168.83.15 and 192.168.162.67 are up and running, since these machines sent RESET packets back to stealth.mappem.com to terminate the connection. Again the ACK number is identical for each of the SYN-ACK packets.

Pattern 26:

```
06:40:12.807419 stealth.mappem.com.113 > 172.21.241.67.2472: S 4212553210:4212553210(0) ack 674711610 win 8192
06:40:35.352131 stealth.mappem.com.113 > 192.168.21.22.2482: S  1245810288:1245810288(0) ack 674711610 win 8192
06:41:24.067330 stealth.mappem.com.113 > 172.21.32.83.1004: S 4052190291:4052190291(0) ack 674711610 win 8192
06:42:08.063341 stealth.mappem.com.113 > 192.168.83.15.2039: S 2335925210:2335925210(0) ack 674711610 win 8192
06:42:08.066294 192.168.83.15.2039 > stealth.mappem.com.113: R 674711610:674711610(0) win 0
06:42:14.582943 stealth.mappem.com.113 > 172.21.64.120.2307: S 2718446928:2718446928(0) ack 674711610 win 8192
06:42:35.858350 stealth.mappem.com.113 > 172.21.79.98.2184: S 3902144771:3902144771(0) ack 674711610 win 8192
06:43:46.974062 stealth.mappem.com.113 > 172.21.126.113.2216: S 3728879575:3728879575(0) ack 674711610 win 8192
06:44:09.602803 stealth.mappem.com.113 > 192.168.162.67.2226: S 761493655:761493655(0) ack 674711610 win 8192
06:44:09.607462 192.168.162.67.2226 > stealth.mappem.com.113: R 674711610:674711610(0) win 0
```

Pattern 27 shows a scan utilizing DNS query responses. Said another way, these are responses to questions that were never posed. As with the RESET scan, there is no stimulus for these connections, and we see the intermediate router responding to stealth.com with host unreachable messages. For this example, the domain traffic is UDP, and the DNS service-specific information has been omitted from the data trace.

Pattern 27:

```
05:55:36.515566 stealth.com.domain > 172.29.63.63.20479: udp
06:46:18.542999 stealth.com.domain > 192.168.160.240.12793: udp
07:36:32.713298 stealth.com.domain > 172.29.185.48.54358: udp
07:57:01.634613 stealth.com.domain > 254.242.221.165.13043: udp
07:57:01.635032 router.mynet.net > stealth.com: icmp: host 254.242.221.165 unreachable
09:55:28.728984 stealth.com.domain > 192.168.203.163.15253: udp
10:38:53.862779 stealth.com.domain > 192.168.126.131.39915: udp
10:38:53.863158 router.mynet.net > stealth.com: icmp: host 192.168.126.131 unreachable
10:40:37.513176 stealth.com.domain > 192.168.151.126.19038: udp
```

10:40:37.513566 router.mynet.net > stealth.com: icmp: host 192.168.151.126 unreachable
10:44:28.462431 stealth.com.domain > 172.29.96.220.8479: udp
11:35:40.489103 stealth.com.domain > 192.168.7.246.44451: udp
11:35:40.489523 router.mynet.net > stealth.com: icmp: host 192.168.7.246 unreachable

Pattern 28 shows a potential denial of service attack against a border router by using icmp echo request packets that are fragmented in 1480 byte chunks. These same packets with the same IP identification numbers are sent repeatedly and directed to against a network address. These are directed against the network address. In this case, the border router actually blocks the fragmented icmp echo requests from reaching the internal subnets. All packets have the same fragment ID and the fragmented packets are sent repeatedly. The denial of service occurs when the router for which the traffic is destined becomes overwhelmed while trying to reassemble the fragments. When the IP was blocked at the external router, the denial of service stopped and the router returned to normal functioning.

Pattern 28:

12:01:12.150572 DOS.org > x.x.x.0: (frag 54050:1480@4440+)
12:01:17.560572 DOS.org > x.x.x.0: (frag 54050:1480@2960+)
12:01:17.570572 DOS.org > x.x.x.0: (frag 54050:1480@4440+)
12:01:22.200572 DOS.org > x.x.x.0: (frag 54050:1480@1480+)
12:01:22.210572 DOS.org > x.x.x.0: (frag 54050:1480@2960+)
12:01:22.220572 DOS.org > x.x.x.0: (frag 54050:1480@4440+)
12:01:22.230572 DOS.org > x.x.x.0: (frag 54050:1480@5920+)
12:01:27.240572 DOS.org > x.x.x.0: (frag 54050:1480@2960+)
12:01:27.250572 DOS.org > x.x.x.0: (frag 54050:1480@5920+)
12:01:37.230572 DOS.org > x.x.x.0: (frag 54050:1480@1480+)
12:01:37.240572 DOS.org > x.x.x.0: (frag 54050:1480@2960+)
12:01:37.240572 DOS.org > x.x.x.0: (frag 54050:1480@4440+)
12:01:37.250572 DOS.org > x.x.x.0: (frag 54050:1480@5920+)
12:01:42.300572 DOS.org > x.x.x.0: (frag 54050:1480@1480+)

The filter that snagged this detection is the same IP filter that comes with SHADOW to detects fragments. The second filter was provided courtesy of Vicki Irwin/Cisco and detects bits turned on in the two high order bits of the reserved flag fields in tcp.

(tcp[13] & 0xc0 != 0)

APPENDIX A
Building an Intrusion Detection System using RedHat Linux 5.1

This documentation shows how we built an intrusion detection system (sensor and analysis system) using RedHat Linux 5.1 as our UNIX operating system and Gateway 2000 Pentium systems. We came across several complications that took a little bit of research, experimentation, and perseverance to overcome. We have included tips where we could, however, every site and system is different and will have its own unique idiosyncrasies. Note: There are several ways to install UNIX software; the procedures in this appendix vary slightly from the installation instructions in Section 3, but still provide valuable insight into a typical installation.

Step 1 Prerequisites

Action 1.1 Legal Review - Get Permission: Self Explanatory

Action 1.2 Modifying UNIX files: Self Explanatory; Quite familiar with UNIX

Action 1.3 Preparing for Installation

(a) Make a list of Hardware specifications. You will need this information later for Linux configuration. The following is a list of Hardware and Software that we used:

HW:	SW:
Pentium II 350 MHZ	RedHat Linux 5.1 (came with Win95 pre-loaded)
128MB RAM	RedHat Linux Patches
9GB SCSI Hard Drive	CID Code
STB Velocity 128bit 8MB Video Card	Libpcap
Generic Multi-Sync Monitor	Tcpdump
	Tcpslice
	Secure Shell

(b) Boot machine under Windows and check and record IRQ settings for all hardware peripherals.

(c) You may run into IRQ conflicts as Windows allows for IRQ sharing, however Linux does not.

(d) If you do happen to find a shared IRQ, one solution is to physically move the network card or the SCSI card to another slot.

Step 2 Acquiring the SHADOW software

Action 2.1 Obtain the SHADOW tar files - completed

Action 2.2 Obtain Secure Shell - completed.

Action 2.3 Download Apache Web Server - skipped; using the Apache version provided in the SHADOW tar set

NOTE:
Steps 3 and 4 were performed concurrently by this site

Step 3 Configuring the Sensor Systems

Step 4 Configuring the Analysis Systems

Action 3.1 and Action 4.1 Obtain computer running UNIX - completed; using Gateway 2000 Pentiums and Red Hat Linux v5.1

Action 3.2 and Action 4.1 Install and Secure the Operating System

(a) Boot machine with RedHat Linux 5.1 boot disk.

(b) Go through initial steps and select install from CDROM (if you are going to install from CD).

(c) Partition Disk. (Our disk contained 9GB of space. For different amounts of space, you will have to partition your disk appropriately)

1. Select Disk Druid.
2. Delete existing partition(s).
3. For the sensor, make 3 partitions using the add option as follows:

mount point	size	type
	127MB	Linux Swap
/	2000MB	Linux Native
/LOG	Rest of disk	Linux Native

4. For the analysis system make 5 partitions using the add option as follows:

mount point	size	type
	127MB	Linux Swap
/	2000MB	Linux Native
/LOG	5059MB	Linux Native
/tmp	512MB	Linux Native
/usr/local	1000MB	Linux Native

Note:
The Linux Boot partition can't be > 2GB (2000MB). Also, you might need to play with the sizes to make them fit.

5. Once done with disk partitions, select OK to move on.

(d) In the next steps, you can select an option to check for bad blocks. We selected this option on the swap partition only, as the other partitions were too big and would ve taken too much time.

(e) Select the components that you will need. This will depend on your site s particular needs. For the sensor, choose very few components. The sensor is outside your firewall, so you don t need to provide hackers with more utilities than you need.

(f) Select type of mouse.

(g) Select video card.

(h) Select monitor and don t probe.

(i) Select Video memory (8MB).

(j) Select no clockchip setting.

(k) Select Video modes:

8bit	16bit	24bit
(x)1280x1024	(x)1280x1024	(x)1280x1024

(l) Configure LAN settings:

1. Enter IP, Netmask, Primary Name Server, Default Gateway, Hostname, Domain Name, Secondary Name Server, Tertiary Name Server

(m) Format Machine time.

(n) Select services to start initially. (Depends on site)

(o) Create Boot disk. This should complete the initial RedHat Linux installation. You now have a working Linux operating system, however improvements need to be made as follows.

(p) Download and install latest patches.

1. Download patches from: http://www.redhat.com to directory /usr/local/archives/rpm (You will probably need to create this directory.)
2. The glint patch is needed before the Package Manager in X Windows can be used.
3. From /usr/local/archives/rpm enter: rpm -Uvh glint*
4. From the X Windows Control Panel, select Package Manager.
5. Select configure and enter path to look for packages (/usr/local/archives/rpm).
6. Select Available to see which packages are not installed.
7. Select packages to install or upgrade.

(q) To use the screensaver and Xlock functions in X Windows, the following packages need to be installed (These can likely be found on the RedHat Linux CDROM):

1. xscreensaver*
2. fortune-mod*
3. xlockmore*

(r) Reboot

(s) Build Custom Kernel. This will allow you to build a smaller kernel with only the components that you need:

1. Instructions are also in the RedHat Linux Manual.
2. Make sure the kernel-headers* and kernel-source* rpm files are upgraded and installed, and perform the next commands as root:
3. cd /usr/src/linux
4. make mrproper
5. make xconfig

(t) Select options based on what your requirements are.

1. make dep
2. make clean
3. make boot
4. make modules
5. rm -rf /lib/modules/2.0.34-0.6-old (2.0.34xx may be different depending on the version of RedHat Linux you are using)
6. mv /lib/modules/2.0.34-0.6 /lib/modules/2.0.34-0.6-old
7. make modules_install
8. mv /boot/vmlinuz-2.0.34-0.6 /boot/vmlinuz-2.0.34-0.6.old
9. cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz-2.0.34-0.6
10. edit /etc/lilo.conf - enter kernel data
11. /sbin/lilo
12. reboot
13. The machine should boot up in the new customized kernel. If you want a choice of booting up in the new kernel or the old kernel press the Tab button when the machine gets to the Lilo: prompt.

(u) Configure security parameters. It is recommended that you set options for your machine to ignore incoming ICMP packets, except from specified sources. In order to do this, the following commands should be entered as root:

- ¥ ipwadm -I -a allow -P icmp -s 192.168.1.0/24 -W eth0 (Instead of 192.168.1.0/24, enter the source(s) from which you will accept ICMP packets)
- ¥ ipwadm -I -a deny -P icmp -s 0.0.0.0/0 -W eth0
- ¥ Run pwconv to incorporate SHADOW passwords (you might need to run this after every password change.)
- ¥ Disable all inetd.conf services.

(v) Linux Installation is now complete.

Action 3.3 and Action 4.2 Unpack the SHADOW tar file: see below

(a) SHADOW code, libpcap, tcpdump, tcpslice, and SSH should have been downloaded to /usr/local/archives/tar (You will probably need to create this directory).

(b) SHADOW code installation:

1. mkdir /usr/local/logger
2. Move SHADOW code to this directory.
3. untar enter: tar xvfz cid*

Action 3.4 and Action 4.2 Build libpcap: see below

(a) libpcap installation

1. uncompress libpcap*
2. tar xf libpcap*
3. cd libpcap-0.4a6
4. ./configure
5. make
6. make install
7. make install-man (you will probably need to mkdir /usr/local/include and mkdir /usr/local/include/net)
8. make install-incl (only for libpcap)

Action 3.5 and Action 4.2 Build tcpdump (also Action 4.4 Build tcpslice for analysis systems)

(a) tcpdump and tcpslice installation

1. repeat steps 1 - 7 for tcpdump and tcpslice.

(a) Secure Shell installation

1. mkdir /usr/local/archives/tar/tmp
2. Move SSH files to this directory.
3. From /usr/local/archives/tar/tmp enter: tar xvfz ssh-1.2.25_linux.install.tar.gz
4. sh -x install _ linux.sh

The remaining Actions were performed to configure the sensor and analysis systems:
Action 3.7 - Modify variables.ph
Action 3.8 - Review activities of tcpdump.driver.pl
Action 3.9 - Test the tcpdump.driver.pl script
Action 3.10 - Configure cron
Action 4.5 - Install Apache Web Server,
Action 4.6 - Configure SHADOW Analyzer
Action 4.7 - Test fetchem.pl
Action 4.8 - Edit cron

Testing was simple:

(a) To use the analyzer, point your browser to: http://your.analysis.system

Congratulations! You now have a working Intrusion Detection System.

Disclaimer

Throughout the document, the examples are based on actual logged network activity. However, in each example, the hostnames and IP addresses have been changed to protect the privacy of the individuals involved. The hostnames used in this document are used for illustrative purposes only, and are not meant to pertain to any individual or organization in existence.

It is very important to observe what you see often and be able to detect the smallest deviance from the ordinary. The examples given above are only a few of the many types of Attacks or probes you may see as you track your network data. Although you may have a lot of data to examine, you must be careful to observe the indications of an attack and be able to spot such an instance even if it is hiding in the middle of a huge amount of data. Again, this is where the one_day_pat.pl may become quite useful. Remember, attackers are persistent and always trying new methods to gain access to your systems, therefore it is vital that you be aware of anything unusual.

INDEX

INDEX