

Preparing Solaris for a Firewall

Firewalls are one of the fastest growing technical tools in the field of information security. However, a firewall is only as secure as the operating system it resides upon. This article will take a step by step look at how you can best armor your Solaris box, both Sparc and x86. These steps can apply to any situation, however I will be using Check Point FireWall-1 on Solaris 2.6 as an example. At the end of this article is a script that you can download that will automate most of the armoring process, to include implementing TCP Wrappers.

Installation

The best place to start in armoring your system is at the beginning, OS installation. Since this is your firewall, you cannot trust any previous installations. You want to start with a clean installation, where you can guarantee the system integrity.

Place your system in an isolated network. At no time do you want to connect your unprotected system to an active network nor the Internet, exposing the system to a possible compromise. I personally witnessed a newly installed system scanned and rooted within 15 minutes of connecting to the Internet. To get critical files and patches later, you will need a second box that acts as a go between. This second box will download files from the Internet, then connect to your isolated, configuration "network" to transfer critical files.

Once you have placed your future firewall box in an isolated network, you are ready to begin. The first step is selecting what OS package to load. The idea is to load the minimum installation, while maintaining maximum efficiency. The less software that resides on the box, the fewer potential security exploits or holes. I recommend Core installation. I prefer Core because this is the absolute minimum installation, creating a more secure operating system. For the truly paranoid, I have created three checklists on how to modify the Core installation. One checklist for Solaris 2.6 and FW-1 4.0, the second checklist for Solaris 2.7 and FW-1 4.1, and the third checklist Solaris 8 and FW-1 4.1. The third checklist is still in beta, as Solaris 8 is not supported by CheckPoint. If you require a GUI, need additional functionality, or are new to Solaris, then you may want to consider the End User installation. Anything above the End User package, such as Developer, is adding useless but potentially exploitable software. Be sure to add the "On-Line Manual Pages" during the install process. I find these extremely useful, while adding little risk to your system. For more information on building a minimal installation, check out Solaris Minimization for Security.

During the installation process, you will be asked to partition your system. I've never really understood Sun's love for making various partitions. You always end up making the partitions too small and running out of room later.

I always like to make root as big as possible and just throw everything in there, then you do not run out of room. However, we do need several partitions to protect the root drive. If we were to fill the root partition with data, such as logging or email, we would cause a denial of service, potentially crashing the system.

Therefore, I always recommend a separate partition for /var, this is where all the system logging and email goes. By isolating the /var partition, you protect your root partition from overflowing. I've found 400 MB to be more than enough for /var. You may also consider making a separate partition for the firewall logging and /usr. If you create a separate partition for /usr, you can mount it read only, protecting the binaries from modification. For Checkpoint Firewall 1, all logging by default happens in /etc/fw/log (/var/opt/CKPfw/log for ver 4.0). Many Solaris systems have two or more drives, such as the Ultra 10 or 2 IDE drives for an x86. If you are not mirroring the second drive, make it the partition for all the firewall logging. Once again, this protects all the other partitions in case the firewall logging floods the drive. With such a setup, your partitions would look as follows:

- / - everything else
- /var - 400 MB
- swap - 256 MB (or normally 2x amount of RAM)
- /etc/fw/log - 2nd drive (for CP FW-1 ver 3.0x)
- /var/opt/CKPfw/log - 2nd drive (for CP FW-1 ver 4.0x)
- /var/opt/CPfw1-41/log - 2nd drive (for CP FW-1 ver 4.1x)

Once the system has rebooted after the installation, be sure to install the recommended patch cluster from Sun. Be sure to use your go between box to get the patches, the firewall box should always remain on an isolated network. Patches are CRITICAL to maintaining a secure firewall and should be updated at least once a week. BUGTRAQ is an excellent source for following the latest bugs and exploits.

Eliminating Services

Once you have loaded the installation package, patches, and rebooted, we are now ready to armor the operating system. Armoring consists mainly of turning off services, adding logging, tweaking several files, and TCP Wrappers.

First we will begin with turning off services.

By default, Solaris is a powerful operating system that executes many useful services. However, most of these services are unneeded and pose a potential security risk for a firewall. The first place to start is /etc/inetd.conf.

This file specifies which services the /usr/sbin/inetd daemon will listen for. By default, /etc/inetd.conf is configured for 35 services, you only need two, ftp and telnet. You eliminate the remaining unnecessary services by commenting them out (example). This is critical, as many of the services

run by inetd pose serious security threats, such as rexd. Confirm what you have commented out with the following command (this will show you all the services that were left uncommented)

```
#grep -v "^#" /etc/inetd.conf
```

The next place to start is /etc/rc2.d and /etc/rc3.d. Here you will find startup scripts launched by the init process. Many of these are not needed. To stop a script from starting during the boot process, replace the capital S with a small s. That way you can easily start the script again just by replacing the small s with a capital S. The following scripts are not needed and pose serious security threats to your system.

/etc/rc2.d

S73nfs.client - used for NFS mounting a system. A firewall should never mount another file system.

S74autofs - used for automounting, once again, a firewall should never mount another file system.

S80lp - used for printing, your firewall should never need to print.

S88sendmail - listens for incoming email. Your system can still send mail (such as alerts) with this disabled.

S71rpc - portmapper daemon, a highly insecure service (required if you are running CDE).

S99dtlogin - CDE daemon, starts CDE by default

/etc/rc3.d

S15nfs.server - used to share file systems, a bad idea for firewalls.

S76snmpdx - snmp daemon

Running any GUI (CDE or OpenWindows) is not a good idea. Only run a GUI when it is absolutely required. You can disable CDE, the default GUI in Solaris 2.6, with the S99dtlogin startup script (replace the capital S with a small s). To get an idea of how many ports and services CDE requires, type the following command when it is running.

```
ps -aef | wc -l
```

Once you are done with the installation and have turned off S99dtlogin and S71rpc (required to run CDE), type the command again and compare how the number of services have decreased. The fewer services running, the better.

For those of you who installed the Core installation, this is not an issue, as the GUI is not installed.

Logging and Tweaking

Once you have eliminated as many services as possible, we want to enable logging. Most system logging occurs in /var/adm. We want to add two additional log files there, sulog and loginlog. /var/adm/sulog logs all su attempts, both successful and failed. This allows you to monitor who is attempting to gain root access on your system. /var/adm/loginlog logs consecutive failed login attempts. When a user attempts to login 5 times, and all 5 attempts fail, this is logged. To enable the files, just touch the files /var/adm/loginlog and /var/adm/sulog. Ensure both files are chmod 640, as they contain sensitive information.

Next comes tweaking. This involves various file administration. The first thing we want to do is create the file `/etc/issue`. This file is an ASCII text banner that appears for all telnet logins (example A). This legal warning will appear whenever someone attempts to login to your system.

We also want to create the file `/etc/ftpusers` (example B). Any account listed in this file cannot ftp to the system. This restricts common system accounts, such as root or bin, from attempting ftp sessions. The easiest way to create this file is the command

```
cat /etc/passwd | cut -f1 -d: > /etc/ftpusers
```

Ensure that any accounts that need to ftp to the firewall are NOT in the file `/etc/ftpusers`.

Also, ensure that root cannot telnet to the system. This forces users to login to the system as themselves and then su to root. This is a system default, but always confirm this in the file `/etc/default/login`, where console is left uncommented (example C).

Last, I like to eliminate the telnet OS banner and create a separate banner for ftp. For telnet, this is easily done by creating the file `/etc/default/telnetd` and adding the statement

```
BANNER="" # Eliminates the "SunOS 5.6" banner for Telnet
```

For ftp, this is easily done by creating the file `/etc/default/ftpd` and adding the statement

```
BANNER="WARNING:Authorized use only" # Warning banner for ftp.
```

Connecting to the Firewall

It is critical that you develop a secured, controlled way to connect to the firewall. Often, you need remote access to your firewall for administration or the uploading of files, these communications need to be secured I will discuss two options here, ssh and TCP Wrappers.

I prefer ssh, as it encrypts all communication between you and the firewall. TCP Wrappers will NOT protect your network traffic from sniffing. Users can still capture all of your keystrokes (including passwords) on the network. If you are concerned about users capturing communications to your firewall, I recommend you replace telnet/ftp with ssh. ssh will encrypt all communications to your firewall, allowing you both to upload files and administer the firewall in a secure manner. ssh is similar to TCP wrappers in that it has its own layer of logging, and can limit what systems can connect to it. For more information on ssh, you can find ssh here. including source for both ssh clients and server daemon. I recommend you use ssh version 1.2.7, as version 2.x has a limiting license. For 95/NT users, I highly recommend SecureCRT as a ssh client.

TCP Wrappers, while it does not encrypt, it does log and control who can access your system. It is a binary that wraps itself around inetd services, such as telnet or ftp. With TCP Wrappers, the system launches the wrapper

for inetd connections, logs all attempts and then verifies the attempt against an access control list. If the connection is permitted, TCP Wrappers hands the connection to the proper binary, such as telnet. If the connection is rejected by the access control list, then the connection is dropped.

Many of you may be wondering why would a firewall need TCP Wrappers, the firewall does all that for you. The answers are simple. First, in case the firewall is compromised or crashes, TCP Wrappers offer a second layer of defense. Second, and just as important, TCP Wrappers protect against Firewall misconfigurations. I have often seen firewalls misconfigured, especially in VPN situations, allowing unauthorized users access to the firewall. Third, TCP Wrappers add a second layer of logging, verifying other system logs.

You can get TCP Wrappers from Wietse Venema's Website. Once again, be sure to use your go between system to retrieve and compile TCP Wrappers. We do not want any compilers on the Firewall and we want to protect the armored Solaris box within its isolated network. Once downloaded, be sure to review the README file first, it is an excellent introduction to TCP Wrappers. Two options I recommend when compiling TCP Wrappers. First, go with paranoid, as this does a reverse lookup for all connections. Second, use the advance configuration, which is actually quite simple. This configuration keeps all the binaries in their original locations, which may be critical for future patches.

Implementing TCP Wrappers will involve editing several files (these examples are based on the advance configuration). First, once compiled, the tcpd binary will be installed in the /usr/local/bin directory. Second, the file /etc/inetd.conf must be configured for which services are to be wrapped (example D). Third, /etc/syslog.conf must be edited for logging tcpd (example E), be sure to touch the file /var/adm/tcpdlog . Last, the access control lists must be created, /etc/hosts.allow and /etc/hosts.deny (example F).

Once all the proper files have been edited and are in place, restart /usr/bin/inetd with kill -HUP. This will restart the daemon with TCP Wrappers in place. Be sure to verify both your ACLs and logging before finishing.

For the Truly Paranoid

I consider the measures discussed above absolutely essential. By following these steps, you have greatly improved your system's security, congratulations! Unfortunately, your system is not 100% secure, nor will it ever be. So, for the truly paranoid, I have added some additional steps you can take.

First we will create the wheel group. The wheel group is a group of select individuals that can execute powerful commands, such as /usr/bin/su. By

limiting the people the can access these commands, you enhance the system security. To create the group, vi the file /etc/group, create the group wheel, and add the system admins to the group. Then identify critical system binaries, such as /usr/bin/su. Change the group ownership to wheel, and the permissions to owner and group executable only (be sure to maintain the suid or guid bit for specific binaries). For /usr/bin/su, the commands would be:

```
/usr/bin/chgrp wheel /usr/bin/su
```

```
/usr/bin/chmod 4750 /usr/bin/su
```

* Note: (Don't forget, for su there is actually another binary in /sbin. For 2.6, this is called /sbin/su.static This is the same thing as /usr/bin/su, however the libraries are statically linked, hence the larger file size. Don't forget to change this file also).

Second, we will lock down the files .rhosts, .netrc, and /etc/hosts.equiv.

The r commands use these files to access systems. To lock them down, touch the files, then change the permissions to zero, locking them down. This way no one can create or alter the files. For example,

```
/usr/bin/touch /.rhosts /.netrc /etc/hosts.equiv
```

```
/usr/bin/chmod 0 /.rhosts /.netrc /etc/hosts.equiv
```

Also, we want to set the TCP initial sequence number generation parameters. By truly randomizing the initial sequence number of all TCP connections, we protect the system against session hijacking and ip spoofing. This is done by setting TCP_STRONG_ISS=2 in the file /etc/default/inetinit (example G). By default, the system installs with a setting of 1, which is not as secure.

To protect against possible buffer overflow (or stack smashing) attacks, add the following to lines to /etc/system.

```
set noexec_user_stack=1
```

```
set noexec_user_stack_log=1
```

Next, we make some modifications to the IP module. Add these commands to one of your start up scripts. For detailed information on ndd and tuning ip modules for security, check out Network Settings for Security.

```
### Set kernel parameters for /dev/ip
```

```
ndd -set /dev/ip ip_respond_to_echo_broadcast 0
```

```
ndd -set /dev/ip ip_forward_directed_broadcasts 0
```

```
ndd -set /dev/ip ip_respond_to_timestamp 0
```

```
ndd -set /dev/ip ip_respond_to_timestamp_broadcast 0
```

```
ndd -set /dev/ip ip_forward_src_routed 0
```

```
ndd -set /dev/ip ip_ignore_redirect 1
```

Last thing I like to do is eliminate as many suid root binaries as possible. suid root binaries pose a high risk, as vulnerable versions can be used to gain root. Since this is a dedicated system with few accounts, most of the suid binaries can be disabled or removed. To find all suid root binaries, run the following command on your system.

```
find / -type f -perm -4000 -exec ls -l {} \; | tee -a /var/tmp/suid.txt
```

Once you have identified all of the suid root binaries, you can remove most of them by changing the permissions to '555', or deleting the binaries

entirely. For example, I eliminated the suid bit on the following binaries from a Core installation of Solaris 2.7.

Conclusion

We have covered some of the more basic steps involved in armoring a Solaris box. The key to a secure system is having the minimal software installed, with protection in layers, such as TCP Wrappers. There are many additional steps that can be taken, such as sudo (allows a system administrator to give limited root privileges to user and log their activities), tripwire (monitor changes in system binaries), and swatch (automated log monitoring and alerts). Remember, no system is truly 100% secure. However, with the steps outlined above, you greatly reduce the security risks.

For more information on how to better armor your Solaris system, check out Sun Microsystems blueprint pages, specifically Solaris Security

For the truly secure, I HIGHLY recommend you check out Brad Powell's armoring script Titan. This professional tool is far more powerful and modular than what I have presented here and documents security in far greater detail. Also, you may want to check out YASSP, Yet Another Security Solaris Package.

Downloads.

To save you the time and trouble, I have created a script file that will do everything we have discussed in this article. The script file will go through your Solaris system and make all the above changes, first backing up any changed files. The script will also implement TCP wrappers for you. This script detects what processor you are using (Sparc or x86) and what version (2.5.1, 2.6, 2.7, and 2.8) and makes the proper changes. I recommend this script for new installs only. Send comments or recommendations to <mailto:lance@honeynet.org>

Download armor-1.3.1.tar.Z

I used compress instead of gzip, since uncompress come with the Solaris distribution.

MD5 Checksum for armor-1.3.1.tar.Z = 45009a639877c7c4015564be97af74fa

Author's bio

Lance Spitzner is currently an active member of the Honeynet Project. He enjoys learning by blowing up systems in his home lab. Before this, he was an Tanker in the Rapid Deployment Force, where he blew up things of a different nature. You can reach him at [mailto: lance@honeynet.org](mailto:lance@honeynet.org) .