

# FOCUS on Sun: Hardening Solaris - Creating a Diamond in the Rough Part Two

Hal Flynn 2000-11-01

## . Introduction

In our last article in this series, we discussed planning for building a **Diamond in the Rough**. We discussed an inventory of needs, and why it is vital to our business plan and server design. We also discussed creating a raw inventory of services.

In this article, we'll discuss refining our raw inventory through organization. We'll point out the benefits of organizing this piece of material. We'll also talk about creating a server blueprint, and the benefits of having one.

Finally, we'll close with using our blueprint to create a hardened system. This system will have the ability to withstand most network threats, while still fulfilling it's mission.

## . Organization

After gathering the information, we should organize our data to create a list. Taking such an approach does several things. First, it allows turn three documents into one, thus simplifying our information. Second, it allows us to extract the pertinent information into an easily readable format, filtering out the distractions.

We can organize using a number of methods. We can use *StarOffice* or *MS Excel* to create a spreadsheet, or we can do it by hand. Without regard to the tools used, the ultimate goal is to create a file that's easily readable and clearly indicates what ports have services running on them.

Using the lists supplied by netstat, rpcinfo, and nmap, we can now make a complete list from the information we've gathered. As the anchor output, we'll use nmap's output, as it tends to be the most complete and esthetic. From there, we'll fill in the pertinent information from the other scans.

The end result should look something like this:

```
7/tcp      open      echo
9/tcp      open      discard
13/tcp     open      daytime
19/tcp     open      chargen
21/tcp     open      ftp
23/tcp     open      telnet
```

|           |      |                 |           |
|-----------|------|-----------------|-----------|
| 25/tcp    | open | smtp            |           |
| 37/tcp    | open | time            |           |
| 42/udp    | open | nameserver      |           |
| 79/tcp    | open | finger          |           |
| 111/tcp   | open | rpcbind         | 100000    |
| 161/udp   | open | snmp            |           |
| 177/udp   | open | xmcp            |           |
| 512/tcp   | open | exec            |           |
| 513/tcp   | open | login           |           |
| 514/tcp   | open | shell           |           |
| 515/tcp   | open | printer         |           |
| 517/udp   | open | talk            |           |
| 520/udp   | open | route           |           |
| 540/tcp   | open | uucp            |           |
| 4045/tcp  | open | lockd           | 100021    |
| 6112/tcp  | open | dtspc           |           |
| 7100/tcp  | open | font-service    |           |
| 32771/tcp | open | status          | 100024    |
| 32772/tcp | open | rusersd         | 100002    |
| 32772/udp | open | sadmin          | 100232    |
| 32773/tcp | open | ttldbserverd    | 100083    |
| 32773/udp | open | rquotad         | 100011    |
| 32774/tcp | open | kcms_server     | 100221    |
| 32774/udp | open | status          | 100133    |
| 32775/tcp | open | cachefsd        | 100235    |
| 32775/udp | open | rusersd         | 100002    |
| 32776/tcp | open | dmispd          | 805306368 |
| 32776/udp | open | sprayd          | 100012    |
| 32777/tcp | open | snmpXdmid       | 100249    |
| 32777/udp | open | rwalld          | 100008    |
| 32778/tcp | open | sometimes-rpc19 |           |
| 32778/udp | open | rstatd          | 100001    |
| 32779/udp | open | cmsd            | 100068    |
| 32786/udp | open | sometimes-rpc26 |           |

As you can see, we've taken the pertinent information from the three files, and merged them into one easily readable file. In the left hand column, we have two vital pieces of information. First, we have the service port number. This is coupled with the protocol the service is using. In the second to the right hand column, we have the name of the service. The far right hand column contains the rpc

service number, identifying the version numbers of those services which are rpc based.

Note that we've saved the **Open** field from the nmap scan. This is an important piece of information, as it tells port is open to the world, or is **Filtered**, as would have been indicated in a situation where the service scanned was either wrapped with TCP Wrappers or protected by a firewall. This field helps clarify the status of the services offered by the machine. Therefore, we retain it.

## • **Determining the Vital Services**

With one complete reference, it is now easier to see what services are enabled, as well as determine the necessary ones. Using our **Inventory of Needs** as a reference, we can start evaluating services and their necessity within our infrastructure.

Enterprise data centers, and the needs within them, tend to be diverse. While some servers are dedicated to serving multiple applications and needs, others are allocated a few specific tasks, such as serving an Oracle Database, or acting as a mail relay for the enterprise mail system. Whatever purposes a system serves, it should be with the smallest offering of services possible.

For our example, the server in our theoretical data center is a mail relay named athens. A machine serving a purpose such as this can do without many of the services that run on a stock installation of Solaris 8. Things such as telnet, rshell, printer, and even X can be shutdown.

Keeping this design in mind, we can take our Inventory of Needs and start designing our system around this plan. We can stop all services with the exception of our **Mail Transport Agent**. In addition, it's likely we want to add an enhanced service or two. We'd more than likely want to add a cryptographic remote login facility to the system. However, we should first look at building a blueprint of our future server.

## • **Blueprinting**

We'll not dabble long in blueprinting. The concept is simple. Building a blueprint is nothing more putting the ideal design specifications of the server on paper. One reason for doing this is that it provides the ability to share the design with staff and management for precedent of design, as well as feedback. Although seemingly trivial, in companies where there are either numerous staff members involved in the building and maintenance of machines, or where the pace is rapid, it is all too easy to deviate from the original design specifications. Documenting them provides reference and formality.

Another reason for drafting this blueprint is to provide information for both system maintenance logs, and the *Site Security Manager*. System maintenance logs should be maintained on each server in the datacenter to keep a running chronology of server events. These logs provide future reference

for system analysis and incident tracking. Whether the log is something as informal as a note pad and a pencil, or as formal as a three-ring binder to retain printed copies of maintenance records which are generated from template files, the goal is to keep a running account of all server activity. Drafting the blueprint and storing it in the system maintenance log gives technical staff a quickly accessible reference. This can be handy for a simple question, or major crisis.

A good Site Security Manager maintains a record of each individual machine on the network. This record contains things such as each individual service the machine is offering, and a history of security incidents. The Site Security Manager also ensures network security scans are performed on a weekly basis, and compares the results with those of the predefined system specifications. The blueprint is a critical piece of information for the Security Manager. Having it readily available to compare with scan results can make the difference between a rapid response and low-impact cleanup process, or delayed response, potentially high profile incident, and expensive cleanup process.

A good blueprint should be simple and direct. Nothing more than a list of what services will be running on the machine. Let's look at an example of this using **athens**, the fictitious mail relay server. Athens needs a minimal amount of services, including sendmail and syslog. In a typical data center environment, we would also want a means of secure remote communication, such as *SSH*. We will for now use telnet, however.

An example blueprint should look something like this:

```
Host: Athens      IP: 192.168.1.2      Netmask: 255.255.255.0

23/tcp           open      telnet
25/tcp           open      smtp
514/udp          open      syslog
```

As one can see, the blueprint is small, simple, and easy to read. There's no mistake as to what services should be running, what ports they should be running on, what protocols they're using, and whether they're filtered or not.

## • From Paper to Production

With our current server design in blueprints, we proceed with shutting down all unneeded services. We'll begin with those in `inetd.conf`.

## • inetd Services

There are various approaches to disabling the services started via `inetd`. Some administrators prefer

to copy the original `inetd.conf` to a different location, and replace it with a new configuration file containing only the pertinent service entries. Another approach is to keep the current configuration in place, and comment out everything not needed. This is strictly a matter of personal preference. For this document, we will use the latter of methods.

From `inetd.conf`, the following services can be disabled:

|                          |                        |                                      |
|--------------------------|------------------------|--------------------------------------|
| <code>echo</code>        | <code>7/tcp</code>     |                                      |
| <code>echo</code>        | <code>7/udp</code>     |                                      |
| <code>discard</code>     | <code>9/tcp</code>     |                                      |
| <code>discard</code>     | <code>9/udp</code>     |                                      |
| <code>daytime</code>     | <code>13/tcp</code>    |                                      |
| <code>daytime</code>     | <code>13/udp</code>    |                                      |
| <code>chargen</code>     | <code>19/tcp</code>    |                                      |
| <code>chargen</code>     | <code>19/udp</code>    |                                      |
| <code>ftp</code>         | <code>21/tcp</code>    |                                      |
| <code>telnet</code>      | <code>23/tcp</code>    |                                      |
| <code>nameserver</code>  | <code>42/udp</code>    |                                      |
| <code>finger</code>      | <code>79/tcp</code>    |                                      |
| <code>exec</code>        | <code>512/tcp</code>   |                                      |
| <code>biff</code>        | <code>512/udp</code>   | (also known as <code>comsat</code> ) |
| <code>login</code>       | <code>513/tcp</code>   |                                      |
| <code>shell</code>       | <code>514/tcp</code>   |                                      |
| <code>printer</code>     | <code>515/tcp</code>   |                                      |
| <code>talk</code>        | <code>517/udp</code>   |                                      |
| <code>uucp</code>        | <code>540/tcp</code>   |                                      |
| <code>dtspc</code>       | <code>6112/tcp</code>  |                                      |
| <code>fs</code>          | <code>7100/tcp</code>  | (font service)                       |
| <code>rusersd</code>     | <code>32772/tcp</code> | (ephemeral)                          |
| <code>sadmin</code>      | <code>32772/udp</code> | (ephemeral)                          |
| <code>ttdbserverd</code> | <code>32773/tcp</code> | (ephemeral)                          |
| <code>rquotad</code>     | <code>32773/udp</code> | (ephemeral)                          |
| <code>kcms_server</code> | <code>32774/tcp</code> | (ephemeral)                          |
| <code>cachefs</code>     | <code>32775/tcp</code> | (ephemeral)                          |
| <code>rusersd</code>     | <code>32775/udp</code> | (ephemeral)                          |
| <code>sprayd</code>      | <code>32776/udp</code> | (ephemeral)                          |
| <code>rwalld</code>      | <code>32777/udp</code> | (ephemeral)                          |
| <code>rstatd</code>      | <code>32778/udp</code> | (ephemeral)                          |
| <code>cmsd</code>        | <code>32779/udp</code> | (ephemeral)                          |

These services are enabled and started by inetd in a default configuration. The **inetd.conf** configuration file contains entries enabling more than the above shown services. However, services such as gssd and amserv require additional configuration, and are not started in a stock installation.

The right column contains the notation **ephemeral** next to some services. These services do not have a well known port number, and do not run on a dedicated port. Instead, these services start a boot time, and run between port ranges 32000 and 33000. The majority of the ephemeral services mentioned above are rpc based.

We comment out all entries in the /etc/inetd.conf file, with the exception of telnetd. This will stop all of the inetd controlled services after inetd is sent the HUP signal, or the system is rebooted. Commenting out all of these services will additionally affect the window manager. Therefore, this task most safely performed at the console via the command line interface.

After commenting everything out of our inetd.conf (except telnet) and sending the HUP signal to inetd, we run a nmap and rpcinfo -p against athens again. Here are the combined results of our scans results:

```
hal@doomgate:~> nmap -sS -sU -O -oN athens athens
```

```
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
```

```
Interesting ports on athens.mrhal.com (192.168.1.2):
```

```
(The 3064 ports scanned but not shown below are in state: closed)
```

| Port      | State | Service         |                  |
|-----------|-------|-----------------|------------------|
| 23/tcp    | open  | telnet          |                  |
| 111/tcp   | open  | sunrpc          | 100000           |
| 111/udp   | open  | sunrpc          | 100000           |
| 161/udp   | open  | snmp            |                  |
| 177/udp   | open  | xdmcp           |                  |
| 514/udp   | open  | syslog          |                  |
| 4045/tcp  | open  | lockd           | 100021           |
| 4045/udp  | open  | lockd           | 100021           |
| 6000/tcp  | open  | X11             |                  |
| 32771/tcp | open  | sometimes-rpc5  | 100133/100024    |
| 32771/udp | open  | sometimes-rpc6  |                  |
| 32772/tcp | open  | sometimes-rpc7  | 805306368        |
| 32772/udp | open  | sometimes-rpc8  | 100133/100024    |
| 32773/tcp | open  | sometimes-rpc9  | 100249           |
| 32774/tcp | open  | sometimes-rpc11 |                  |
| 32776/udp | open  | sometimes-rpc16 | 300598/805306368 |

```
32777/udp  open          sometimes-rpc18  100249
32778/udp  open          sometimes-rpc20
```

TCP Sequence Prediction: Class=random positive increments

Difficulty=73573 (Worthy challenge)

Remote operating system guess: Sun Solaris 8 early acces beta through actual release

Nmap run completed -- 1 IP address (1 host up) scanned in 205 seconds

As we can see, our efforts have been fruit bearing. Let's now move on to eliminating services started the in rc scripts.

## • **init Started Services**

Now that we've eliminated the majority of the services, we must fine tune the system via the **system initializations scripts**. We first start in the /etc/rc3.d directory.

In the /etc/rc3.d directory, we find the following:

```
# pwd
/etc/rc3.d
# ls
README          S15nfs.server  S50apache      S76snmpdx      S77dmi
#
```

None of these scripts start services necessary to the mail relay server. However, we want to retain these scripts in the event of needing them in the future as company needs change. Therefore, we move them all to filenames with a prefix of "no," to prevent them from running at system boot. This also is a good way to let others responsible for the system know these services have been disabled on purpose. We move the following files to the following extensions:

/etc/rc3.d:

```
S15nfs.server  => noS15nfs.server
S50apache      => noS50apache
S76snmpdx     => noS76snmpdx
S77dmi        => noS77dmi
```

Next, we perform the same task in /etc/rc2.d. We move the following files to eliminate the rest of

the listening services on athens:

```
/etc/rc2.d:
```

```
S71rpc          => noS71rpc
S73nfs.client   => noS73nfs.client
S74autofs       => noS74autofs
S99dtlogin      => noS99dtlogin
```

With these adjustments made, scanning our host now looks like this:

```
hal@doomgate:~> nmap -sS -sU -oN athens athens
```

```
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
```

```
Interesting ports on athens.mrhal.com (192.168.1.2):
```

```
(The 3079 ports scanned but not shown below are in state: closed)
```

| Port    | State | Service |
|---------|-------|---------|
| 23/tcp  | open  | telnet  |
| 25/tcp  | open  | smtp    |
| 514/udp | open  | syslog  |

```
Nmap run completed -- 1 IP address (1 host up) scanned in 180 seconds
```

We've now reached our desired configuration.

## • Some Final Adjustments

Before we launch our system into the unknown, there are a few things we can do to provide a higher level of security.

1) We can set our system up to generate sequence numbers with more security. This makes TCP sequence number guessing attacks a bit more difficult. Solaris 8 is preset to a `TCP_STRONG_ISS` level of 1. For true randomness, this level can be set to 2. This is, however, at a network performance degradation of around two percent. This option can be specified in `/etc/default/inetinit`. Use your judgment in specifying this option.

2) We can install the [OpenSSL](#) and [OpenSSH](#) toolkits. For more information, there is an article available here that discusses the build of OpenSSL and OpenSSH. While the example system is not Solaris (the example uses Linux), the concepts are essentially the same.

These options are not mandatory, although they are ideal for a strong security configuration.

## • Conclusion

In this document, we have discussed using our inventories to create a "Diamond in the Rough." We have discussed designing and building a server that can withstand the majority of network intrusion attempts. We discussed using blueprints, and walked through the steps necessary to design a host that is secure and accomplishes the overall mission.

We discussed the organization aspects of creating a network hardened server, as well as the technical aspects. And finally, we mentioned some extra configurations that can help secure a "Diamond in the Rough."

## Relevant Links

[Diamond in the Rough Pt. I](#)

*Hal Flynn*

[Intro to IPFilter I](#)

*Jeremy Rauch*

[Intro to IPFilter II](#)

*Jeremy Rauch*

[Clear Text I](#)

*Hal Flynn*

[OpenSSH Project](#)

*OpenSSH*

[Privacy Statement](#)

Copyright 2006, SecurityFocus