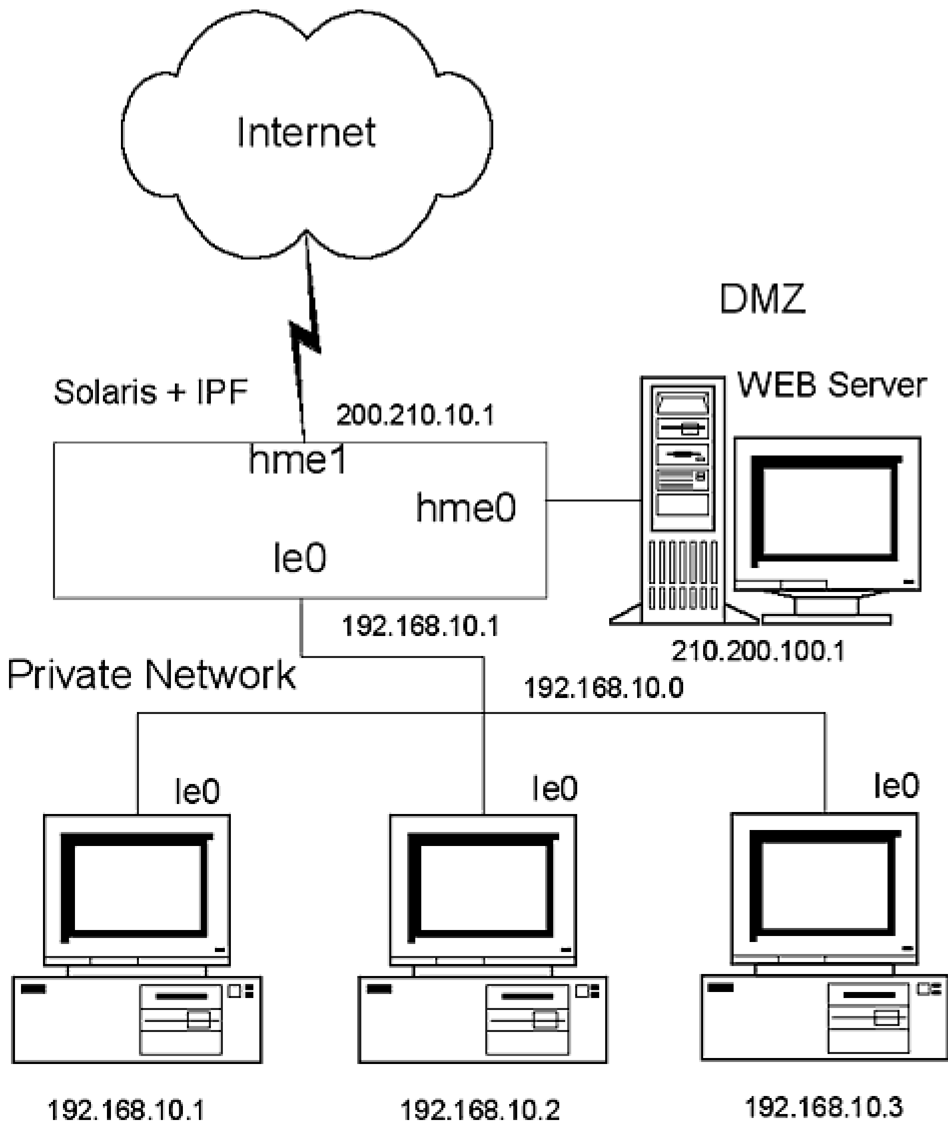
Turn a Solaris Box into a Packet-filtering Firewall

by Boris Loza, PhD, CISSP

Today with the Internet growing so rapidly, it is almost impossible to avoid being connected online. But connecting to the Internet, or to any network that we don't trust, requires making security preparations. In this article, we're going to show you how to create a packet-filtering firewall with a Solaris box using a simple network configuration like the one shown in **Figure A**.

Figure A: *This is a simple network configuration appropriate for a small office.*



Nowadays, probably everybody has heard of firewalls and what they do. But, maybe not everybody knows that a Solaris box can be easily turned into an inexpensive, packet-filtering firewall. Before going into detail as to how to do this, let's go over what a firewall is.

What's a firewall?

Simply put, a firewall is a device used to control network traffic. It prevents outsiders from gaining access to your network and stops unauthorized connections from the inside.

In order to accomplish these tasks, a firewall needs to know at least the following information: a source IP address, a destination IP address, a source port, a destination port, and flags (TCP only). This information can be retrieved from a network packet header. Depending on how this information compares to the firewall access control list, the packet is either allowed to pass or it's dropped. A firewall that looks at this minimal amount of information is a static packet filter.

In addition to the information used by a static, packet-filtering firewall, a dynamic packet filter maintains a connection table in order to monitor the state of a communication session. This type of a firewall is able to remember previous communication packet exchanges and decide (based on this additional information) whether the session is allowed or denied.

Another firewall type is a proxy or an application firewall. With a proxy firewall source and destination, systems will never actually connect to each other. Instead, they're talking to the proxy for all connection requests. In this case, you must create a proxy application for each networked service (one for ftp, another for telnet, another for HTTP, and so forth). Even though a proxy firewall is application aware, it can provide more security than any other type of firewall.

In this article, we'll show you how to install, configure, and run a freely available firewall—Internet Packet Filter (IPF). IPF is a hybrid firewall, which is a dynamic packet filter with some proxy capabilities. Written by Darren Reed, IPF is capable of selectively filtering any protocol listed in the `/etc/protocols` file. You load IPF into the Solaris kernel between the data link (the actual NIC) and the network (for example, IP) layers (layers 2 and 3 of OSI model). By inspecting at this layer, IPF intercepts and inspects all inbound and outbound packets on all interfaces.

Preparing the system

In order to function as a firewall, your machine should have at least two network interfaces in addition to `lo0` (loopback). If you've met these requirements, begin system preparation by installing only those packages that are absolutely required by the OS. Sun allows you only to install as a minimum the core meta-cluster. Because this cluster contains packages that aren't really needed for a properly secured system, delete the following:

- Files that present specific security concerns, such as `snoop, finger, rlogin, rdist,` etc.

- Files that are part of a package that's deemed unnecessary for a firewall, such as NIS, UUCP, and the Solaris Desktop/CDE environment.
- Files that will be replaced by a more secure version of that service, such as `in.telnetd` and `in.ftpd` (which will be replaced with SSH) or aren't necessary for a secure build (for example, `nfs.client`, `nfs.server, rpc, autofs, sendmail, in.routd,` etc.).

Now, install the latest recommended patch cluster and Y2K patch cluster and comment out all unnecessary services in the `/etc/inetd.conf` file. Next, install the following third-party software tools, which are required to complete a secure build: SSH, S3, Tripwire, and fix-mode. Ensure that all interfaces on the machine have unique MAC addresses, including the virtual address, by executing the following command:

```
#eeprom local-mac-address?=true
```

Ensure that interface load balancing is not enabled (Solaris 2.6 only). All outgoing data from the same source must have the same source IP address:

```
#/usr/sbin/ndd -set
=> /dev/ip ip_enable_group_ifs 0
```

Now, enable IP forwarding with the following command:

```
#/usr/sbin/ndd -set ip_forwarding 1
```

Optionally, you can prepare two separate Solaris boxes to act as one firewall. Then, install a High Availability system between these boxes and run in asymmetric mode.

Installing IPF

Now you need to install the IPF. You'll find the latest version of the IPF source code (currently version 3.3.6) at:

ftp://coombs.anu.edu.au/pub/net/kernel

Once you've downloaded the IPF, it's easy to install. To do so, simply type the following command:

```
#make solaris
```

This will work for both Solaris 2.3-6 and Solaris 2.4-6x86. Just make sure to use the Sun's native `make` utility—`/usr/ccs/bin/make`, as the GNU make won't work!

Once you've successfully compiled the IP Filter, change the directory to SunOS5 and type:

```
#make package
```

This generates a packaged version of the IPF and you'll be asked by a `pkgadd` utility if you would like to install it. If you decide not to install IPF at this point, you can do it later on by issuing the following command:

```
Pkgadd -d .
```

If you decide to proceed with an installation, IPF installs the package and loads the IPF kernel module from the `/usr/kernel/drv` directory. In addition to the kernel modules, IPF installs files into the `/usr/sbin`, `/opt/ipf`, `/etc/opt/ipf,` and `/etc/init.d` directories.

To check if the installation completed, issue the following commands:

```
#modinfo | grep ipf
#pkginfo  ipf
```

If you see the information about IP Filter, you're all set.

Creating IPF filters

We'll base our examples on the simple network topology depicted in **Figure A**. The `/etc/opt/ipf/ipf.config` is a configuration file in which we'll put all our filter rules. The format of this file is simple; there's only one rule per line, and the pound sign (#) indicates a comment. The rules syntax is simple and easy to understand. Suppose we'd like to allow HTTP connections from the Internet only to our Web server with an IP address of 210.200.100.1. In this case, the ipc.config file will look like this:

```
pass in log quick on hme1 from any
=> to 210.200.100.1/32 port = 80

block in log quick on hme1 from
=> any to any port = 80
```

The first line is a rule that says "Pass and log all incoming traffic (`in`) that goes through the hme1 interface (`on hme1`) from any source to the target 210.200.100.1 (`/32` is a netmask), and that's addressed to the HTTP (`port = 80`)." The keyword `quick` tells IPF to take this action immediately and not to bother checking the rest of the filter table. To enable these rules, type:

```
#/etc/init.d/ipfboot stop
#/etc/init.d/ipfboot start
```

To allow only established connections to pass through, we could modify our `ipf.conf file` as follows:

```
block in on hme1
=> pass out quick on hme1 proto tcp/udp
=> from 210.200.100.0/24 to any
=> keep state
pass out quick on hme1 proto icmp
=> from 210.200.100.0/24 to any
=> keep state
pass in quick on hme1 proto tcp
=> from any to 210.200.100.0/24
=> port = 80
```

The workings of the keep state rule set are much like the workings of the saying "do not speak until spoken to." Communication into the firewall is just not permitted (except on port 80). See **Listing A** for more examples of firewall rules that are worth adding into the configuration file.

Listing A: *Examples of firewall rules*

```
#Block packets that are too short
#to contain a complete header.
#Block packets with IP options:
block in log quick from any to any with short
block in log quick from any to any with ipopts
#Block broadcasts:
block in quick from any to 0.0.0.255/0x000000ff
#Block Anti-spoofing attacks:
block in log quick on hme1 from 127.0.0.0/8 to any
block in log quick on hme1 from 10.0.0.0/8 to any
block in log quick on hme1 from 192.168.0.0/16 to any
block in log quick on hme1 from 172.16.0.0/12 to any
```

As you can see from all these examples, IPF has many different options and various rule-setting syntaxes. For detailed information about IPF's options, consult `IPF(5)'s` man pages. You can also examine the examples rules found in `/opt/ipf/examples`.

IPF and NAT

NAT (Network Address Translation) converts all the hidden source addresses and port numbers behind the firewall to the IP addresses of the firewall running NAT. Our private network in **Figure A** uses non-routable IP addresses. These ranges are defined by the Network Information Center for private use only (RFC1869—"Address Allocation for Private Internets"). So NAT effectively allows systems on this network to share a single registered IP address on the Internet. This allows all of the machines behind the firewall to use Internet services such as http, ftp, telnet, and email.

In order to enable NAT, we have to create a file called `/etc/opt/ipf/nat.conf`. By adding the following line into the `nat.conf` file for our private network, we will say

```
map hme1 192.168.10.0/24 ->
=> 200.210.100.1/32
```

Whenever a packet goes across the hme1 interface with a source address matching the 192.168.10.0/24 it will be rewritten within the IP stack such that its source address is 200.210.100.1. Then the packet will be sent to its original destination. The system also keeps a list of which translated connections are in progress so that it can perform the reverse and remap the response (which will be directed to 200.210.100.1) to the internal host that really generated the packet.

If you use a dynamically assigned IP addresses, you have to rewrite this rule:

```
map hme1 192.168.10.0/24 -> 0/32
```

In this case you'll have to run `ipf -y` to refresh the address if you get disconnected. But what happens to the source port when the mapping occurs? The packet's source port is changed to the first available port on the firewall. If you don't desire this behavior, and wish to limit the amount of ports that the NAT system can consume, we can add the `portmap` keyword:

```
map hme1 192.168.1.0/24 -> 0/32
=> portmap tcp/udp 10000:30000
```

Now the NAT translates all connections (which can be TCP, UDP, or TCP/UDP) into the port range of 10000 to 30000. For more information about NAT, refer to the manual pages or the "IP Filter Based Firewalls HOWTO" found at http://www.obfuscation.org/ipf/ipf-howto.txt.

IPF, TCP Wrapper, and Nmap

If you install IPF on a single host with one network card, you can still benefit from the services that IPF provides. If you're a longtime reader of Inside Solaris, you should be familiar with TCP Wrapper. Now let's see what advantages IPF has over TCP Wrapper.

First, TCP Wrapper cannot protect services that start at boot time and run until system shutdown, like `sendmail` and `httpd`. IPF doesn't have this limitation. In addition to TCP, it can also block UDP, ICMP, and other protocols. Assume that we installed IPF on 192.168.10.1 in order to protect this host from any other hosts on a private network, as shown in **Figure A**. Let's see how we can block various TCP and UDP services:

```
#Allow telnet to pass from
#192.168.10.3 only:
pass in quick on le0 proto
=> tcp from 192.168.10.3
=> to 192.168.10.1 port = telnet
=> keep state
pass in log quick on le0
=> proto tcp from 192.168.10.3
=> port > 1023 to 192.168.10.1
=> port = telnet keep state

block in log quick on le0 proto tcp
=> from any to any port = telnet
```

If we would like to stop `rlogin` and `rsh` services, we have to add the following rules:

```
block in proto tcp from any
=> to any port 512 >< 514
```

This filter will prevent any incoming TCP connection to ports 513 (`rlogin`) and 514 (`rsh`). To block and log any inbound UDP packets that are going to the NFS port 2049, you can write the following rule:

```
block in log on le0 proto udp
=> from any to any port = 2049
```

In addition, IPF can fool `nmap` and other Internet scanners. To misguide the attacker into believing that there are no services to break into, we can create our own TCP response. When a service isn't running on a UNIX system, it normally replies with an RST (Reset) packet. When blocking a TCP packet, IPF can actually return an RST to the origin by using the `return-rst` keyword:

```
block return-rst in log quick
=> from any to 192.168.10.1 proto
=> tcp port = telnet
block in log quick on le0
pass in all
```

If somebody sends a packet to a UDP port, we can use the `return-icmp` keyword to send a reply:

```
block return-icmp (port-unr) in log
=> quick on le0 proto udp from any to
=> 192.168.10.1/32 port = 111
```

In this case, a `port-unreachable` message will return as if no service is listening on the port in question.

Unfortunately, if you don't block a service completely, a SYN type of attack cannot be defeated with IPF. For this reason, you might want to log all initial SYN packets for further investigation:

```
log in on le0 proto tcp from
=> any to any flags S/SA
```

Tools

IPF comes with various tools that help you perform comprehensive monitoring and debugging. Because of the limited space of this article, we won't go into details about all of them.

First, we'd like to mention the `ipmon` utility. `Ipmon` watches the packet log as created with the `log` keyword in our rules. Running in the background with the `-o` option, `ipmon` will send packet information through `syslogd`. The default facility for this is local0. If you add an entry into the `/etc/syslog.conf` file (for example, `local0.info /var/adm/ipf.log`), you can use the `ipf.log` file as a pattern-matching source for any host-based intrusion detection system.

If you decide to watch events as they happen in real-time, run `ipmon` with `-o I` options. Also, `ipmon` lets you look at the NAT table in action (`ipmon -o N`) and the state table (`ipmon  o S`).

To see a snapshot of how IPF is doing, use the `ipfstat` utility. You'll see a statistic of how many packets have been passed or blocked, if they were logged or not, how many state entries have been made, and so on. It's also capable of showing your current rule list.

With `ipftest,` you can test your rule file. It reads in a filter file and then applies sample IP packets to it. This allows for testing of the filter list and examination of how a packet is passed along through it. And `mkfilters` will suggest that you use a set of filter rules to add routes to back this up. For more information about these and other utilities, consult IPF's man pages.

Further reading

Since it isn't possible to cover all firewall/IP filter aspects in one article, an excellent book about firewalls in general is Firewalls and Internet Security: Repelling the Wily Hacker, by William R. Cheswick and Steven M. Bellovin. One of the best references for how to secure a Solaris system is Securing Solaris: Step-by-Step, which you can find at http://www.sansstore.org/. Further, you can find a list of free and commercial firewall products in Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network. And of course, for more information about IPF, you can check the IP Filter Based Firewalls HOWTO.