Configuring Netfilter/iptables with Shorewall

Setting the Table

When users think about their workstations at home, they often forget about security. But danger is out there, waiting to pounce on the unsuspecting. Shorewall helps everyday Linux users keep the intruders away.

BY JAMES MOHR

The only way to make your system completely safe from attack is never to connect to the Internet. Whenever you open the door to go out, you open the door for a potential hacker to get it.

Some users have the mistaken belief that intruders only attack high-profile sites and would not be interested in a personal home workstation. The truth is that every single computer on the Internet is in danger of attack – even computers with dial-up connections. In many cases, these attacks are blind, brute force attacks, where the attacker tries a long list of known security holes.

To foil the attackers, you could simply disable all inbound ports, which essentially makes your computer invisible to the world. However, when you need to (or even just want to) provided services from your computer, you need a different form of protection.

Even the smallest businesses need the protection a firewall provides. Most users cannot afford the thousands of dollars for a commercial firewall product. Fortunately, there are open source solutions that offer the necessary protection. One such solution is Shorewall.

Behind the Scenes

Shorewall is the common name of the Shoreline Firewall. From a user's perspective, Shorewall is a set of easy-to-configure files that are used to configure Netfilter [1]. Netfilter is a feature of the Linux 2.4.x and 2.6.x kernels that allows kernel modules to access the network protocol stack at various places. The kernel module then can do almost anything with the network packet, including simply dropping it or even manipulating it.

Netfilter also supports the older ipchains, the packet filter facility in 2.2 kernels. Netfilter needs to be explicitly set to "ipchains compatibility mode" for it to work with ipchains.

You can download the latest version of Shorewall from the Shorewall Web site [2] as either source or as an RPM package. Note that the RPM version has not been tested with every distribution, although it has been tested with major distributions like SuSE, Redhat, and Mandrake. If you are uncertain, check the Shorewall site.

In order for Shorewall to work, you also need the iptables and iproute/iproute2 packages. These packages are provided by default in most distributions, so it shouldn't be a problem. The reason you need iptables is that Shorewall is not really the firewall itself. That is, Shorewall is not responsible for checking, filtering, and managing packets. Shorewall simply takes its configuration files and uses the *iptables* command to load them into the kernel.

Because *iptables* assumes the task of manipulating tables within the kernel, Shorewall is no longer needed once you run it. You can actually see what is being done by taking a look at the Shorewall program itself. No, you don't need to dig through a lot of source. The shorewall program (typically, */sbin/shorewall*) is a shell script.

So that iptables will know what to do, you need to tell the kernel what the rules are. So-called *rulesets* are defined within iptables and consist of one connection and a number of "classifiers." This determines if a particular connection is allowed, if and how the packet should be manipulated or redirected, and so on.

This concept is basically the same in all firewall software and can be employed on networks with dozens or even hundreds of computers, although it is likely that with that many computers talking to each other, you would probably want to break down your network into multiple segments. Each segment also could be managed by its own Shorewall firewall.

One thing to note is that you do not necessarily need to have a dedicated computer just for your firewall. Although this is common practice (and generally a good idea), home users probably don't have the space to set up extra computers for each specific function. If your workstation is connecting directly to the Internet, you could add a firewall directly to your workstation.

My network consists of one computer with a DSL card that connects to the Internet, one Windows XP machine, and another Linux computer. Each has a different purpose, and I allow different connections to and from these machines.

As simple as my network configuration is, I only have to change a couple of

Listing 1: Sample *zones* file

```
01 #ZONE DISPLAY COMMENT
02 net Net          the Internet zone
03 loc Local        the local network
04 fw  FW  the firewall
05 #LAST LINE -- ADD YOUR ENTRIES BEFORE THIS ONE -- DO NOT REMOVE
```

the configuration files in order to use Shorewall. Because this is a common (as well as simple) configuration, it is a good place to start.

Basic Configuration

The primary configuration file is */etc/shorewall/shorewall.conf*. *shorewall.conf* lets you configure everything from startup behavior to shutdown behavior. Although you can set a lot of different values in this file, I have not yet found a reason to change any of the settings on my system.

You may already be familiar with the term "segment" to refer to specific portions of a network. Shorewall uses the term "zone." In my configuration, I have four zones: *fw* (the firewall itself, my workstation), *net* (the Internet) and *loc* (the local network).

Zone names must be short (5 characters or fewer) and can contain letters or numbers. Note that you cannot use the special zone *all*. Also, you cannot refer to a zone other than your firewall by the name defined by the FW variable in *shorewall.conf*. This name defaults to *fw*.

Even if you are not providing any services to computers outside of your local network, you still need the Internet zone. Remember that iptable rules are defined by a specific connection, that is, by two end points. One end point is perhaps your workstation, and the other is the Internet. Thus, you need to define the Internet as a specific zone.

Note that these names are just a convention. Although they are the default values that Shorewall itself uses for these zones, you could call them anything you want, provided the naming is consistent through all of the configuration files.

By default, the permissions on the */etc/shorewall* directory are 700, which means only the owner (root) has access. Even read permission could be dangerous, as someone *might* be able to see a hole in the security and exploit it.

The zones are defined in the */etc/shorewall/zones* file. Each entry has three values: zone name (used to reference this zone in the other files), display name (which appears when shorewall is loading the rules), and a comment. Listing 1 shows you a basic *zones* file.

Defining the Paths of Communication

Although we have defined the zones, the firewall still does not know how it communicates with each zone. That is, there is no association between the zone name and the actual network. This is done through the */etc/shorewall/interfaces* file, which consists of four columns: zone, interface, broadcast, and options.

On my system, the *interfaces* file looks like Listing 2.

The zone is simply the name of the zone from your *zones* file. The interface is the name of the network interface. For example, for my DSL connection, which uses the Point-to-Point Protocol (PPP), the interface name is *ppp0*. The interface name for my ethernet card is *eth0*. To find out which interfaces you have, use the command */sbin/ifconfig*. The broadcast column is the broadcast address for the network attached to that interface. As you might guess, the options column specifies any options you want to use.

On my system, I have two entries that look like this:

```
net ppp0 routefilter,norfc1918
loc eth0 detect -
```

As you can see, the Internet zone is connected to the *ppp0* interface and the local zone is connected to the *eth0* interface. Since I have three zones, you may be asking yourself why there is no entry for the firewall zone. Well, quite simply, the firewall zone connects to the other zones through one of the interfaces already specified. So, it might be better to think of the *interface* file as defining which interface the *firewall* uses to talk to the other zones.

In the case of the *ppp0* interface, the broadcast has a hyphen (-). Since a PPP connection doesn't have a broadcast, I could have left this blank. However, since I wanted to specify some additional options, I needed some kind of place holder. Hence, the hyphen. If I had no options, I could have left this blank. However, I like to always include hyphens as a reminder that something is "missing."

The *routefilter* option I specified tells the kernel to reject any package on the given interface that has a source address that would have been routed outbound on a different interface. In this case, if the *ppp0* interface had a packet with an inbound source address that was normally routed *outbound* from the *eth0* interface, it would be dropped. This is referred to as *anti-spoofing*.

The second option, *norfc1918*, tells the kernel not to route addresses specified as "private" in RFC 1918. RFC 1918 lists a number of address ranges that can be used by anyone and *shouldn't* be routed. This option ensures that they aren't. Details of RFC 1918 can be found at [3].

Here I have a dilemma. I want my other computers to be able to access the Internet, but they have RFC 1918 addresses. So how can they reach the Internet? Well, this is something we will get to shortly.

You can configure Shorewall to behave in specific ways based on the zones by setting the "policy" for that zone. Setting the default policy for a zone is done (as you might guess) through the *policy* file. The fields in this file are: client (the source zone), server (the destination zone), policy (what should be done by

Listing 2: Sample *interfaces* file

```
01 #ZONE INTERFACE BROADCAST OPTIONS
02 net     ppp0    -       routefilter,norfc1918
03 loc     eth0    detect   -
04 #LAST LINE -- ADD YOUR ENTRIES BEFORE THIS ONE -- DO NOT REMOVE
```

default), and log level (how much information to record).

Listing 3 shows a sample policy file. Take a look at the last two lines. The first one says that any traffic that has not yet matched and is coming from the Internet is to simply be dropped (ignored).

You have four choices for the connection policy:

ACCEPT – Accept the connection request. DROP – Ignore the connection request. REJECT – Return an appropriate error to the requesting computer. CONTINUE – Allows you to have hosts in multiple zones and apply policies from both zones.

The log level determines how much information is sent to the system logger (syslog). Keep in mind that all of this is done by iptables, which exists in the kernel. That means all of the logging information is sent from the kernel logging facility. All you are doing here is specifying the priority of messages that are logged. (For more details on system logging, check out the syslogd manpage.)

The last two lines of my policy file look like this:

```
net all DROP info
all all REJECT info
```

The first line says any traffic that has not yet matched and is coming from the Internet is to simply be dropped (ignored). However, the remaining traffic is rejected. This may seem a little odd at first, but it makes sense when you consider the difference between reject and drop. When a packet comes from the Internet intended for a specific address or port that is not allowed, I want to *ignore* it. I do *not* want the sender to know what happened. This might give the user a clue as to how to circumvent the security. On the on the hand, packets coming from anywhere else (my workstation or the local machine) are rejected so the client application has some idea of what happened.

Note that we are configuring the *default* behavior in the zone file. What this means is that if you do not define any other specific connections, the behavior in the policy file will apply.

In my *policy* file, I have only this one entry for the net zone. This theoretically would mean that all packets coming

Listing 3: Sample *policy* file

```
01 #SOURCE    DEST      POLICY    LOG LEVEL
02 fw         net       ACCEPT    info
03 fw         loc       ACCEPT    info
04 loc        net       REJECT    info
05 loc        fw      ACCEPT    info
06 net        all       DROP      info
07 all        all       REJECT    info
08 #LAST LINE -- ADD YOUR ENTRIES BEFORE THIS ONE -- DO NOT REMOVE
```

from the Internet are dropped. However, I have a web server on my machine with a lot of reference information I want to be able to access from work. Since this connection goes across the Internet, the entry in the *policy* file would mean that I could not access my web server.

The answer to this is the *rules* file. The *rules* file is the heart of the Shorewall configuration. Here you define a specific connection both in terms of zones, network services (ports), network segments, individual machines, and basically any combination you can think of. When an incoming request arrives, the system first checks the request against connections defined in the *rules* file. If none are found, it uses the default values in the policy file. The columns in the *rules* file are: action to be taken, the source of the request, the destination, the protocol used, the destination port, the source port, and the original destination.

In addition to the actions in the *zone* file, you have a couple of additional actions in the *rules* file. DNAT allows you to do Destination Network Address Translation. With this, requests can be forwarded to different computers and even different ports on those computers. The REDIRECT action will redirect requests to specific ports on the same machine. This is commonly used to redirect HTTP requests to a local proxy (i.e., squid).

For smaller networks that do not want to provide full Internet access, I have found that the REDIRECT action is extremely useful (perhaps only allowing HTTP access). I have any entry that looks like this:

```
REDIRECT loc 3128 tcp www - ➥
!10.2.38.0/24
```

In a nutshell, this entry says that all incoming connections using the TCP protocol and requesting www services (i.e., HTTP) are redirected to port 3128. This is the port on which the Squid proxy server is listening.

Looking at the original destination, we specify an entire class C network using CIDR notation: 10.2.38.0/24. Plus, this entry is preceded by an exclamation mark. As in other contexts, the exclamation mark is used to negate the entry. Therefore, the meaning of the whole line is reversed. That is, all requests *except* those destined for addresses in the 10.2.38.0 network are redirected to Squid. This makes sense because 10.2.38.0 is my local network, and any Web servers on the local network can or should be accessible without having to go through a proxy.

The LOG action will first log the packet and then continue on with the next applicable rule. The QUEUE action is used to forward packets to user-space applications, which manipulate them and return them to the IP stack.

A rule of thumb when configuring your *rules* file is to turn everything *off* > at first, and then turn things back on as you need them. Many people do things in reverse, by starting with a network that is wide open and turning off unneeded services. If you forget something in the first case, it simply means you don't have the access you expect (which is a little inconvenient). In the second case, forgetting something could mean you open yourself up to an attack.

Digging Deeper

The */etc/shorewall/hosts* file allows you to define specific hosts. Usually, there is no reason to add anything into the *hosts* file for smaller networks. The smaller the network, the more likely that all machines in a zone are configured the same way (or more accurately phrased, that all of the machines on a given *interface* have the same configuration).

This may not be the case, and you need different access rules for different machines on the same network. A couple of months ago, my son began to play an Internet roll playing game. This meant he needed more than just HTTP access to the Internet, so I could no longer simply use the redirection to the Squid proxy, as described above.

To provide this access, I defined a specific set of computers as a new zone. So, I first created a new entry in my zone file for this zone, which I called "game." Next, I created an entry in the *hosts* file that contained just my son's computer:

```
game eth0:10.2.38.13
```

But that's not all. Remember that my local network is 10.2.38.0/24. This is one of the "private" networks listed in RFC 1918. Even if I wanted to route this through my DSL card, you can bet money that my ISP is not going to route it. So what could I do?

The solution is something called "IP masquerading." As its name implies, one IP address is "masquerading" for the others. In my case, the IP address on the DSL card (which has a valid Internet address) is masquerading for the IP addresses in the local network. So, I needed to create an entry in the */etc/shorewall/masq* file that looked like this:

```
ppp0 10.2.38.0/24
```

This entry says that all traffic outbound on the *ppp0* interface coming from the class C network 10.2.38.0 is to be masquerade.

At this point, the basic configuration for the masquerading was done. However, next came the *hardest* part. It was not very easy to get information about running this game across a firewall. In fact, I found references that actually said if you want to play, you need to disable all firewalls while you play!

Well, I eventually got it to work. To do so, I needed to set the logging level in the *policy* file to debug and watch the log file for all attempts to connect from my son's machine to *any* machine, then check DNS to see if that was a machine from the game developers. I then added the specific port to my *rules* file.

If I wanted too, I could have added rules for the WWW service (port 80) as well. However, since this was the only thing my son needed other than Web access, I still continue to use the REDIRECT entry for web access.

Troubleshooting and Logging

It is possible (if not likely), that you may have problems the first time you configure Shorewall. So, being able to track down the causes of a problem will be useful.

One useful debugging technique is setting the logging level in the policy file. Setting it to debug gives you a lot more information. Going into detail about the log entries is beyond the scope of this article, but they are fairly easy to figure out, even without knowing all of the

individual entries. For example, take a look at this entry:

```
Nov 1 11:19:32 saturn kernel: ➥
Shorewall:net2all:DROP:IN=ppp0 ➥
OUT= MAC=
SRC=1.2.3.4 DST=10.2.38.11 ➥
LEN=48 TOS=0x00 PREC=0x00 ➥
TTL=116 ID=47048 DF
PROTO=TCP SPT=1 292 DPT=1080 ➥
WINDOW=64240 RES=0x00 SYN URGP=0
```

This is a standard entry from */var/log/messages*, so at the beginning you have the date, the computer name, and the syslog facility (kernel, in this case). Following that is the actual message. You can easily tell that this packet was inbound on the *ppp0* (my DSL card) and the packet was dropped. You can also see that it was using the connection *net2all*, meaning from the Internet to all other zones.

If you look back to the discussion on the *policy* file, you see that we had an entry that said the default behavior was to drop all packets from the Internet to *all* other interfaces. There was a packet trying to reach the destination port (DPT) 1080, the socks port. I don't have anything running on that port, and I certainly did not communicate to anyone that they should try to use that port. Since there is not a "standard" service associated with this port, there is no reason for someone to try to access it from the Internet, so it seems pretty obvious to me that someone was trying to see if they could exploit a Windows bug.

If the remote computer (or even another computer on the same network) continued to try to access various ports on my machine, I might want to "blacklist" them. I would do this by adding the host or network address (in CIDR format) into the */etc/shorewall/ blacklist* file. This means that, regardless of any other entries that might allow them access, the computer or network is specifically denied access.

By default, Shorewall uses the system logging facility, which typically defaults to sending message to */var/log/messages*. Even when I configure *syslogd* to send messages to another file, I find it bothersome to have my firewall logs get mixed up with other kernel messages.

To solve this, you can use the ULOG support in the kernel. Note that ULOG support must be available in your kernel, but this is the default in most newer distributions. However, the *ulogd* package is not always available, so you may have to download it yourself from [4].

Once *ulogd* is configured for your system, you no longer use the *syslogd* logging levels in the policy. Instead, you use the ULOG. ■

INFO

[1] Netfilter Website: *http://www.netfilter.org*

[2] Shorewall Website: *http://www.shorewall.net*

[3] Address Allocation for Private Internets RFC 1918: *http://rfc.net/rfc1918.html*

[4] Ulog: *http://www.ulog.ch/english/index.html*

[5] Squid Proxy: *http://www.squid-cache.org/*