Filtering Bridges

Abstract

Often it is useful to divide one physical network (like an Ethernet) into two separate segments without having to create subnets, and use a router to link them together. The device that connects the two networks in this way is called a bridge. A FreeBSD system with two network interfaces is enough in order to act as a bridge.

A bridge works by scanning the addresses of MAC level (Ethernet addresses) of the devices connected to each of its network interfaces and then forwarding the traffic between the two networks only if the source and the destination are on different segments. Under many points of view a bridge is similar to an Ethernet switch with only two ports.

Table of Contents

1. Why use a filtering bridge?

More and more frequently, thanks to the lowering costs of broad band Internet connections (xDSL) and also because of the reduction of available IPv4 addresses, many companies are connected to the Internet 24 hours on 24 and with few (sometimes not even a power of 2) IP addresses. In these situations it is often desirable to have a firewall that filters incoming and outgoing traffic from and towards Internet, but a packet filtering solution based on router may not be applicable, either due to subnetting issues, the router is owned by the connectivity supplier (ISP), or because it does not support such functionalities. In these scenarios the use of a filtering bridge is highly advised.

A bridge-based firewall can be configured and inserted between the xDSL router and your Ethernet hub/switch without any IP numbering issues.

2. How to Install

Adding bridge functionalities to a FreeBSD system is not difficult. Since 4.5 release it is possible to load such functionalities as modules instead of having to rebuild the kernel, simplifying the procedure a great deal. In the following subsections I will explain both installation ways.

Do not follow both instructions: a procedure *excludes* the other one. Select the best choice according to your needs and abilities.

Before going on, be sure to have at least two Ethernet cards that support the promiscuous mode for both reception and transmission, since they must be able to send Ethernet packets with any address, not just their own. Moreover, to have a good throughput, the cards should be PCI bus mastering cards. The best choices are still the Intel EtherExpress™ Pro, followed by the 3Com® 3c9xx series. To simplify the firewall configuration it may be useful to have two cards of different manufacturers (using different drivers) in order to distinguish clearly which interface is connected to the router and which to the inner network.

2.1. Kernel Configuration

So you have decided to use the older but well tested installation method. To begin, you have to add the following rows to your kernel configuration file:

```
options BRIDGE
options IPFIREWALL
options IPFIREWALL_VERBOSE
```

The first line is to compile the bridge support, the second one is the firewall and the third one is the logging functions of the firewall.

Now it is necessary to build and install the new kernel. You may find detailed instructions in the Building and Installing a Custom Kernel section of the FreeBSD Handbook.

2.2. Modules Loading

If you have chosen to use the new and simpler installation method, the only thing to do now is add the following row to /boot/loader.conf:

```
bridge_load="YES"
```

In this way, during the system startup, the bridge.ko module will be loaded together with the kernel. It is not required to add a similar row for the ipfw.ko module, since it will be loaded automatically after the execution of the steps in the following section.

3. Final Preparation

Before rebooting in order to load the new kernel or the required modules (according to the previously chosen installation method), you have to make some changes to the /etc/rc.conf configuration file. The default rule of the firewall is to reject all IP packets. Initially we will set up an open firewall, in order to verify its operation without any issue related to packet filtering (in case you are going to execute this procedure remotely, such configuration will avoid you to remain isolated from the network). Put these lines in /etc/rc.conf:

```
firewall_enable="YES"
firewall_type="open"
firewall_quiet="YES"
firewall_logging="YES"
```

The first row will enable the firewall (and will load the module ipfw.ko if it is not compiled in the kernel), the second one to set up it in `open` mode (as explained in /etc/rc.firewall), the third one to not show rules loading and the fourth one to enable logging support.

About the configuration of the network interfaces, the most used way is to assign an IP to only one of the network cards, but the bridge will work equally even if both interfaces or none has a configured IP. In the last case (IP-less) the bridge machine will be still more hidden, as inaccessible from the network: to configure it, you have to login from console or through a third network interface separated from the bridge. Sometimes, during the system startup, some programs require network access, say for domain resolution: in this case it is necessary to assign an IP to the external interface (the one connected to Internet, where DNS server resides), since the bridge will be activated at the end of the startup procedure. It means that the fxp0 interface (in our case) must be mentioned in the ifconfig section of the /etc/rc.conf file, while the xl0 is not. Assigning an IP to both the network cards does not make much sense, unless, during the start procedure, applications should access to services on both Ethernet segments.

There is another important thing to know. When running IP over Ethernet, there are actually two Ethernet protocols in use: one is IP, the other is ARP. ARP does the conversion of the IP address of a host into its Ethernet address (MAC layer). In order to allow the communication between two hosts separated by the bridge, it is necessary that the bridge will forward ARP packets. Such protocol is not included in the IP layer, since it exists only with IP over Ethernet. The FreeBSD firewall filters exclusively on the IP layer and therefore all non-IP packets (ARP included) will be forwarded without being filtered, even if the firewall is configured to not permit anything.

Now it is time to reboot the system and use it as before: there will be some new messages about the bridge and the firewall, but the bridge will not be activated and the firewall, being in `open` mode, will not avoid any operations.

If there are any problems, you should sort them out now before proceeding.

4. Enabling the Bridge

At this point, to enable the bridge, you have to execute the following commands (having the shrewdness to replace the names of the two network interfaces fxp0 and xl0 with your own ones):

```
# sysctl net.link.ether.bridge.config=fxp0:0,xl0:0
# sysctl net.link.ether.bridge.ipfw=1
# sysctl net.link.ether.bridge.enable=1
```

The first row specifies which interfaces should be activated by the bridge, the second one will enable the firewall on the bridge and finally the third one will enable the bridge.

At this point you should be able to insert the machine between two sets of hosts without compromising any communication abilities between them. If so, the next step is to add the `net.link.ether.bridge.[blah]=[blah]` portions of these rows to the /etc/sysctl.conf file, in order to have them execute at startup.

5. Configuring The Firewall

Now it is time to create your own file with custom firewall rules, in order to secure the inside network. There will be some complication in doing this because not all of the firewall functionalities are available on bridged packets. Furthermore, there is a difference between the packets that are in the process of being forwarded and packets that are being received by the local machine. In general, incoming packets are run through the firewall only once, not twice as is normally the case; in fact they are filtered only upon receipt, so rules that use `out` or `xmit` will never match. Personally, I use `in via` which is an older syntax, but one that has a sense when you read it. Another limitation is that you are restricted to use only `pass` or `drop` commands for packets filtered by a bridge. Sophisticated things like `divert`, `forward` or `reject` are not available. Such options can still be used, but only on traffic to or from the bridge machine itself (if it has an IP address).

New in FreeBSD 4.0, is the concept of stateful filtering. This is a big improvement for UDP traffic, which typically is a request going out, followed shortly thereafter by a response with the exact same set of IP addresses and port numbers (but with source and destination reversed, of course). For firewalls that have no statekeeping, there is almost no way to deal with this sort of traffic as a single session. But with a firewall that can "remember" an outgoing UDP packet and, for the next few minutes, allow a response, handling UDP services is trivial. The following example shows how to do it. It is possible to do the same thing with TCP packets. This allows you to avoid some denial of service attacks and other nasty tricks, but it also typically makes your state table grow quickly in size.

Let's look at an example setup. Note first that at the top of /etc/rc.firewall there are already standard rules for the loopback interface lo0, so we should not have to care for them anymore. Custom rules should be put in a separate file (say /etc/rc.firewall.local) and loaded at system startup, by modifying the row of /etc/rc.conf where we defined the `open` firewall:

```
firewall_type="/etc/rc.firewall.local"
```

You have to specify the *full* path, otherwise it will not be loaded with the risk to remain isolated from the network.

For our example imagine to have the fxp0 interface connected towards the outside (Internet) and the xl0 towards the inside (LAN). The bridge machine has the IP `1.2.3.4` (it is not possible that your ISP can give you an address quite like this, but for our example it is good).

```
# Things that we have kept state on before get to go through in a hurry
add check-state

# Throw away RFC 1918 networks
add drop all from 10.0.0.0/8 to any in via fxp0
add drop all from 172.16.0.0/12 to any in via fxp0
add drop all from 192.168.0.0/16 to any in via fxp0

# Allow the bridge machine to say anything it wants
# (if the machine is IP-less do not include these rows)
add pass tcp from 1.2.3.4 to any setup keep-state
add pass udp from 1.2.3.4 to any keep-state
add pass ip from 1.2.3.4 to any

# Allow the inside hosts to say anything they want
add pass tcp from any to any in via xl0 setup keep-state
add pass udp from any to any in via xl0 keep-state
add pass ip from any to any in via xl0

# TCP section
# Allow SSH
add pass tcp from any to any 22 in via fxp0 setup keep-state
# Allow SMTP only towards the mail server
add pass tcp from any to relay 25 in via fxp0 setup keep-state
# Allow zone transfers only by the secondary name server [dns2.nic.it]
add pass tcp from 193.205.245.8 to ns 53 in via fxp0 setup keep-state
# Pass ident probes.  It is better than waiting for them to timeout
add pass tcp from any to any 113 in via fxp0 setup keep-state
# Pass the "quarantine" range
add pass tcp from any to any 49152-65535 in via fxp0 setup keep-state

# UDP section
# Allow DNS only towards the name server
add pass udp from any to ns 53 in via fxp0 keep-state
# Pass the "quarantine" range
add pass udp from any to any 49152-65535 in via fxp0 keep-state

# ICMP section
# Pass 'ping'
add pass icmp from any to any icmptypes 8 keep-state
# Pass error messages generated by 'traceroute'
add pass icmp from any to any icmptypes 3
add pass icmp from any to any icmptypes 11

# Everything else is suspect
add drop log all from any to any
```

Those of you who have set up firewalls before may notice some things missing. In particular, there are no anti-spoofing rules, in fact we did *not* add:

```
add deny all from 1.2.3.4/8 to any in via fxp0
```

That is, drop packets that are coming in from the outside claiming to be from our network. This is something that you would commonly do to be sure that someone does not try to evade the packet filter, by generating nefarious packets that look like they are from the inside. The problem with that is that there is *at least* one host on the outside interface that you do not want to ignore: the router. But usually, the ISP anti-spoofs at their router, so we do not need to bother that much.

The last rule seems to be an exact duplicate of the default rule, that is, do not let anything pass that is not specifically allowed. But there is a difference: all suspected traffic will be logged.

There are two rules for passing SMTP and DNS traffic towards the mail server and the name server, if you have them. Obviously the whole rule set should be flavored to personal taste, this is only a specific example (rule format is described accurately in the ipfw(8) man page). Note that in order for "relay" and "ns" to work, name service lookups must work *before* the bridge is enabled. This is an example of making sure that you set the IP on the correct network card. Alternatively it is possible to specify the IP address instead of the host name (required if the machine is IP-less).

People that are used to setting up firewalls are probably also used to either having a `reset` or a `forward` rule for ident packets (TCP port 113). Unfortunately, this is not an applicable option with the bridge, so the best thing is to simply pass them to their destination. As long as that destination machine is not running an ident daemon, this is relatively harmless. The alternative is dropping connections on port 113, which creates some problems with services like IRC (the ident probe must timeout).

The only other thing that is a little weird that you may have noticed is that there is a rule to let the bridge machine speak, and another for internal hosts. Remember that this is because the two sets of traffic will take different paths through the kernel and into the packet filter. The inside net will go through the bridge, while the local machine will use the normal IP stack to speak. Thus the two rules to handle the different cases. The `in via fxp0` rules work for both paths. In general, if you use `in via` rules throughout the filter, you will need to make an exception for locally generated packets, because they did not come in via any of our interfaces.

6. Contributors

Many parts of this article have been taken, updated and adapted from an old text about bridging, edited by Nick Sayer. A pair of inspirations are due to an introduction on bridging by Steve Peterson.

A big thanks to Luigi Rizzo for the implementation of the bridge code in FreeBSD and for the time he has dedicated to me answering all of my related questions.

A thanks goes out also to Tom Rhodes who looked over my job of translation from Italian (the original language of this article) into English.