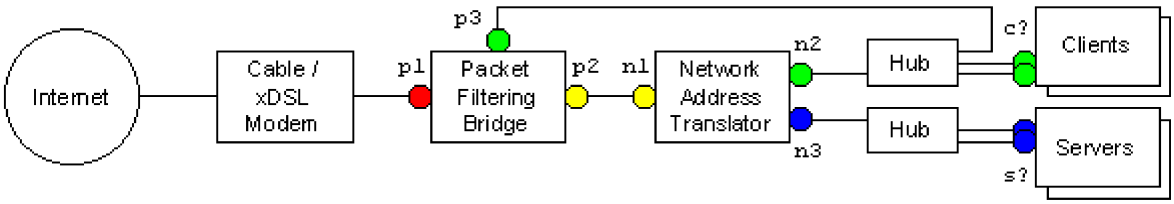
A network setup with FreeBSD and OpenBSD

by Jan Sipke van der Veen (jansipke@yahoo.com)

1. Overview

This article discusses a network setup which might prove useful for people who like to put some extra effort into connecting their machines to the Internet. The goal is to build a secure client and server farm on a single IP address. Next to the clients and servers, the network contains two other components. The first one is a Network Address Translator (NAT), which makes it possible to connect more than one machine to the Internet using just one IP address. The second one is a Packet Filtering Bridge (PFB) that transparently filters IP traffic from and to the Internet.



The PFB machine - running OpenBSD 2.9 - is the only machine connected directly to the Internet. Two of its network interfaces are for the bridge itself (p1 and p2) and one is the management interface (p3). Interface p3 makes it possible for the PFB to output its logs to another server, or for an administrator to log into it to check its logs. This might not seem very secure at first sight, but the interface is actually protected as well as the clients on the LAN.

The NAT machine - running FreeBSD 4.3 - is connected with one interface to the PFB machine (n1). The other two interfaces are connected to the hubs of the clients (n2) and servers (n3). This seperation between a client and a server LAN has at least one obvious advantage. If a weakness is found in one of the services of a server, the clients are still relatively safe from attack. Otherwise, the clients would be on the same network as the compromised server and would have made an easy target for the attacker.

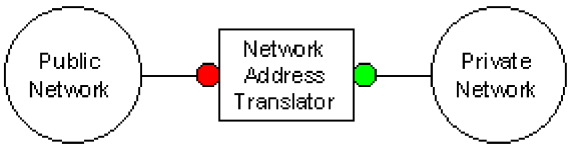
2. Theory

Let's start with some theory about Network Address Translators (NAT) and Packet Filtering

Bridges (PFB).

2.1 Network Address Translator (NAT)

Commonly, your Cable or xDSL ISP assigns you a very limited number of IP addresses (usually just one). You run into trouble if you want to connect more machines to the Internet than the number of IP addresses given to you. NAT is a technology which you can use to solve this problem to some extent. With NAT, multiple internal IP address (on the LAN) are aliased to a single public IP address (on the Internet). Requests from machines on the LAN to machines on the Internet are transformed by the NAT machine. It modifies the source address in the IP packet and sends the request in its name to the machine on the Internet. Once it gets a reply, it modifies the destination address in the IP packet to the client on the LAN.



Typically, a NAT machine has two network interfaces, one on the Internet and one on the LAN. The public IP address on the Internet is assigned to you by your ISP, so you assign this IP address to the interface connected to the public network. But what about the private IP addresses on the LAN? There are some IP addresses which are available for exactly this situation. You can choose between three IP ranges:

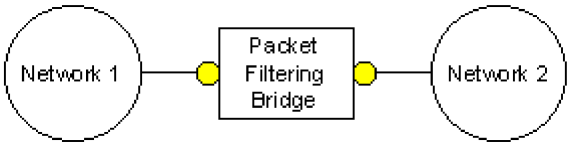
- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Let's say you are using the third IP range. You would then assign one of the IP addresses in this range (e.g. 192.168.0.1) to the interface connected to the private network. Every other machine on this network is assigned one of the remaining addresses in the range. Here's an example of the complete IP configuration of the machines involved:

	IP address	Subnet mask	Gateway
NAT public	216.136.204.21	255.255.248.0	216.136.204.1
NAT private	192.168.0.1	255.255.255.0	No gateway
LAN machine 1	192.168.0.2	255.255.255.0	192.168.0.1
LAN machine 2	192.168.0.3	255.255.255.0	192.168.0.1

2.2 Packet Filtering Bridge (PFB)

A bridge is used to connect two network segments to create a single, larger network. This is normally done to create the appearance of a single network when there are in fact two physically seperate networks. The bridge in this article is not just a simple bridge, but it functions as a packet filter as well. Devices that filter packets have rulesets that tell them which packets should be allowed and which should be blocked. Typically, a firewall is used for this job, but a disadvantage of this setup is, that the firewall is visible on the Internet. This makes it a great target to anyone trying to get into your network. If the firewall gets compromised, all machines within the LAN are in the open, waiting to be compromised as well.



In our case, the packet filtering is done by the bridge. A normal bridge takes all packets from the network on one interface and copies them to the interface on the other network and vice versa. Our PFB examines each of these packets before they are copied to the other interface. It checks its ruleset to see if a packet should be allowed to pass or should be blocked. Unlike the NAT machine, the PFB machine doesn't change anything in the packets themselves.

The IP configuration of the PFB and the machines on both networks is really simple. The PFB is not assigned an IP address at all. This ensures that machines on either side of the PFB are not aware of its existence. Because of this, the machines on both networks should be configured as if the two networks were really one larger one. The cool thing about this, is that the PFB can't be reached from the Internet, making it impossible to attack it on the IP level.

3. Implementation

Now that we've seen the theory, let's shift our attention to the implementation. As mentioned before, we are using FreeBSD 4.3 for the NAT machine and OpenBSD 2.9 for the PFB machine. We won't go into much depth with regard to the client and server machines, although we will discuss the IP settings of these machines.

3.1 Network Address Translator (NAT)

Kernel

We want to be able to divert and filter IP packets. The FreeBSD generic kernel doesn't allow this, so we have to make our own kernel. We do this by making a copy of the generic kernel:

```
cd /usr/src/sys/i386/conf
cp GENERIC NAT
```

Then we add the following lines to the NAT file:

```
options IPDIVERT
options IPFIREWALL
```

Now we can compile our new kernel and install it:

```
config NAT
cd ../../compile/NAT
make depend
make
make install
```

Configuration

We should now edit some configuration files to really enable the capabilities just added to the kernel. The first file is */etc/rc.conf* which contains the network interface configuration as well as the services configuration:

```
hostname="nat.example.com"                              # Host- and domainname
ifconfig_xl0="216.136.204.21"                           # Network interface connected to the F
ifconfig_rl0="inet 192.168.0.1 netmask 255.255.255.0"   # Network interface connected to the (
ifconfig_rl1="inet 10.0.0.1 netmask 255.255.255.0"      # Network interface connected to the s
defaultrouter="216.136.204.1"                           # Default router on network interface

firewall_enable="YES"                                   # Enable firewall capabilities
gateway_enable="YES"                                    # Enable gateway capabilities
natd_enable="YES"                                       # Enable network address translation
natd_interface="xl0"                                    # Set public NAT interface to n1
natd_flags="-f /etc/natd.conf"                          # Set rules file for the NAT daemon
```

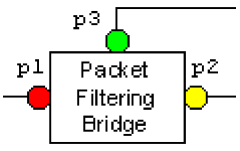
The second file is */etc/natd.conf* which contains the rules for the NAT daemon:

```
redirect_port tcp 10.0.0.3:22 22                        # Redirect SSH packets to the SSH serv
redirect_port tcp 10.0.0.4:80 80                        # Redirect HTTP packets to the HTTP se
```

The third file is */etc/rc.firewall* which contains the rules for the packet filter. In our case we first divert all IP packets to the NAT daemon. After that, we make sure that IP packets can only go where they are supposed to go. Strictly speaking, we don't need the rules below the one that diverts IP packets to the NAT daemon. However, we would otherwise have to allow all IP packets, and that is something we don't like doing:

```
# Flush all previous firewall rules
/sbin/ipfw -f flush

# Divert all IP packets to the NAT daemon
/sbin/ipfw add divert natd ip from any to any via xl0

# Allow all IP packets through the loopback interface by the localhost
/sbin/ipfw add allow ip from 127.0.0.1 to 127.0.0.1 via lo0

# Allow all IP packets through the public interface (n1) from and to this machine
/sbin/ipfw add allow ip from any to 216.136.204.21 in recv xl0
/sbin/ipfw add allow ip from 216.136.204.21 to any out xmit xl0

# Allow all IP packets through the client interface (n2) from and to client machines
/sbin/ipfw add allow ip from 192.168.0.1/24 to any in recv rl0
/sbin/ipfw add allow ip from any to 192.168.0.1/24 out xmit rl0

# Allow all IP packets through the server interface (n3) from and to server machines
/sbin/ipfw add allow ip from 10.0.0.1/24 to any in recv rl1
/sbin/ipfw add allow ip from any to 10.0.0.1/24 out xmit rl1

# Deny all IP packets not allowed until this point
/sbin/ipfw add deny ip from any to any
```

3.2 Packet Filtering Bridge (PFB)



Kernel

We want to be able to divert and filter IP packets. The OpenBSD generic kernel has support for both these features, so there is no need to make our own kernel.

Configuration

The first configuration files to edit are the ones that set the parameters for the network interfaces. File */etc/hostname.ne0* is for the network interface connected to the client LAN (p3):

```
inet 192.168.0.2 255.255.255.0 NONE
```

Files */etc/hostname.ne1* and */etc/hostname.ne2* are for the network interfaces connected to the Cable / xDSL modem and NAT machine respectively (p1 and p2). Because we won't be giving them IP addresses, the content of these files is simply:

```
up
```

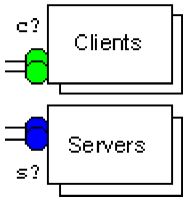
Now we should edit the file */etc/bridgename.bridge0* to enable the bridge at startup:

```
add ne1
add ne2
up
```

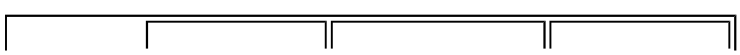
The last configuration file to edit is */etc/ipf.rules* which contains the rules for the packet filter. As with the NAT machine, we make sure that IP packets can only go where they are supposed to go. There is a difference however: we want to protect our network from the Internet. We tell the packet filter that it is okay for the NAT machine to send IP packets to the Internet and receive replies on these packets. We also want to offer SSH and HTTP services to the Internet, so we explicitly open these ports on our bridge:

```
# Allow all IP through the loopback interface by the localhost
pass in quick on lo0 from 127.0.0.1 to 127.0.0.1

# Allow all IP through the LAN interface from and to LAN addresses
pass in quick on ne0 from 192.168.227.0/24 to 192.168.227.2
pass out quick on ne0 from 192.168.227.2 to 192.168.227.0/24

# Allow all IP through the bridge from the NAT machine
pass in quick on ne2 proto tcp/udp from 216.136.204.21 to any keep state
pass in quick on ne2 proto icmp from 216.136.204.21 to any keep state

# Allow SSH and HTTP through the bridge from the Internet to the NAT machine
pass in quick on ne1 proto tcp from any to 216.136.204.21 port = 22 flags S keep state
pass in quick on ne1 proto tcp from any to 216.136.204.21 port = 80 flags S keep state

# Deny IP not allowed until this point
block in quick all
```

3.3 Clients and Servers



To complete our setup we must configure the clients and servers. Our client LAN is connected through a hub to interface n2 of the NAT machine, so all clients should be in the same subnet as n1. For the same reason, all servers should be in the same subnet as n3. The configuration looks like this:



	IP address	Subnet mask	Gateway
Client 1	192.168.0.3	255.255.255.0	192.168.0.1
Client 2	192.168.0.4	255.255.255.0	192.168.0.1
Server 1	10.0.0.3	255.255.255.0	10.0.0.1
Server 2	10.0.0.4	255.255.255.0	10.0.0.1

4. Testing

From one of the client machines we will now start the test of our configuration. Let's see if one of our clients (192.168.0.3) can reach the NAT machine (192.168.0.1):

```
%ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=0.560 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=0.435 ms
...
```

We expect about the same result if we try to reach the NAT machine (10.0.0.1) from one of our servers (10.0.0.3). If this works OK, we should try the PFB. We can do this from the NAT machine by pinging the gateway of our ISP:

```
%ping 216.136.204.1
PING 216.136.204.1 (216.136.204.1): 56 data bytes
64 bytes from 216.136.204.1: icmp_seq=0 ttl=255 time=20.340 ms
64 bytes from 216.136.204.1: icmp_seq=1 ttl=255 time=21.636 ms
...
```

We can repeat this test from one of our clients or servers to test the NAT daemon. Notice that the Time To Live (ttl) value is 1 lower than before. This is because the NAT machine is now a hop the IP packets have to pass to reach the gateway. Our final test for our setup will be the HTTP server, which we will assume is on IP address 10.0.0.4. First, try to fetch an HTML page from the terminal of the NAT machine itself with this URL:

```
http://10.0.0.4/directory/file.html
```

If this succeeds, you can try to fetch the same HTML page from a machine on the Internet with this URL:

```
http://216.136.204.21/directory/file.html
```

The NAT machine should redirect this request to the HTTP server on IP address 10.0.0.4 and give you the same result as the test above.

5. References

FreeBSD

- natd
- rc.conf
- ipfw

OpenBSD
- hostname.*

- bridgename.*
- ipf