Christoph Schneeberger, January 2003

GSEC v 1.4b practical assignement
Option 1

# “Firewalling IPv6 with OpenBSD's pf (packet filter)”

## Table of Contents

# 1 Introduction / Goals

## 1.1 Abstract

This paper will walk the reader through the process of setting up an OpenBSD machine as firewall for IPv4 as well as IPv6, which is described as "dualstacked" configuration within the following paper.

In the first chapter the reader will be familiarized with the reasons behind: why a company, organisation or person would need a dualstacked firewall during transition from IPv4 to IPv6.

The special problems of this scenario to be solved will be described, like the fact that IPv4 is most probably run with NAT/PAT (Network Address Translation / Port Address Translation) but the IPv6 protocol is normally routed into the internal net, therefore the minimum default protection of NAT doesn't apply anymore.

The reader will also be shown and explained the network setup on which this paper is based.

Then the reader will be briefly introduced on how to install and configure an OpenBSD machine for the use as a firewall. This chapter is kept short, since this information is well available all over the web. Nevertheless a few references and recommendations will be given on how to further secure and tighten the machine's installation and configuration.

In the main chapter about configuring the firewall rules itself the reader will be introduced to a relatively easy policy on which the rules will be based. Next, the resulting IPv4 rules will be shown and explained where necessary. Following the IPv6 ruleset for the given policy will be shown and the main differences to the IPv4 ruleset will be explained. In a final step, both rulesets will be combined and the result will be checked with tools available on the Internet.

Finally, some more complex examples are given, i.e. allowing inbound traffic to a dualstacked web- and mailserver as well as letting clients autoconfigure their IPv6 address from our firewall.

The presented solution should give Small Offices to large Enterprises a good start into the IPv6 world without the need of a hard transition.

## 1.2 Introduction

As the Internet grew in size and traffic, a new protocol had to be introduced to fulfill the new upcoming needs of the Internet community. In the past the largest Internet growth was caused by the booming computing industry. This growth is expected to flatten somewhere in the future, but other devices are upcoming which will need a unique internet address too, like i.e. Personal Digital Assistants or Cell Phones as well as other consumer devices like TV sets, washing machines etc.

It might be feasible to satisfy the needs of computer users without IPv6 with the help of techniques like CIDR and NAT.

CIDR[1], Classless Inter-Domain Routing is used to collapse routing information, divide large A- and B-Class networks as well as aggregating routing information on backbones.

NAT (Network Address Translation) / PAT (Port Address Translation). Most often PAT is meant when NAT is said. The NAT/PAT technique allows a site to use RFC1918[2] addresses within it's site and handle all traffic over one registered and globally routed internet address. We will accordingly use the term NAT, even if we do a mixture of NAT and PAT with our firewall.

IPv6 or IPng was formally standardized in November 1994 with RFC1752[3] and a transition of large

1 RFC1519 - Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy
2 RFC1918 - Address Allocation for Private Internets
3 RFC1752 - The Recommendation for the IP Next Generation Protocol

parts of the Internet was believed to happen until 2001. Obviously this hasn't happened yet, at the time of this writing there are still very few ISPs offering native IPv6 services.

This paper should help an IT Manager of a network of small to medium size to make transition as smooth as possible while retaining policy controls on one single machine.

## 1.3 Goals to achieve with this setup

The goals we want to achieve are

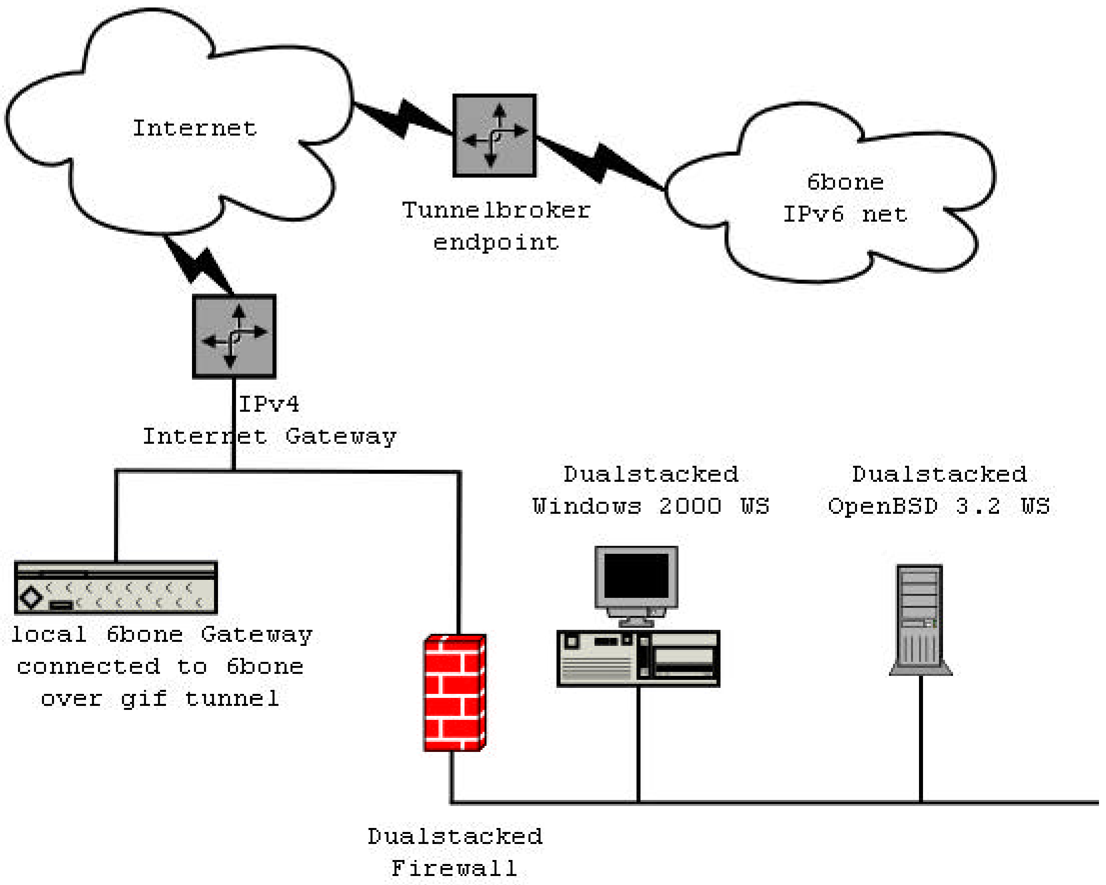
1. Getting routable IPv6 address space into the internal network run with RFC1918 IPv4 addresses.
2. Conserve the same level resp. a similar level of security for both protocol suites
3. Supporting internal machines with IPv4 only, IPv6 only and dualstacked machines at the same time for internet connectivity.

### 1.3.1 Why would somebody want a dualstacked firewall ?

IT managers experimenting with IPv6 while still keeping current IPv4 connections alive. Or people/organizations that do not want or can not put a special testbed in place for IPv6 but still intend to experiment with IPv6.

## 1.4 Description and drawing of the planned setup

The planned setup uses 2 internal machines representing the internal network structure for test purposes. Then there is the firewall itself as well as the 6bone gateway which could also be the same machine as the firewall, but to keep the setup simple and the firewall rules straight the 6bone gif tunnel to our tunnel broker runs on dedicated hardware. The following diagram shows the testbed we will use for our setup:


Internet
Tunnelbroker endpoint
6bone IPv6 net
IPv4 Internet Gateway
Dualstacked Windows 2000 WS
Dualstacked OpenBSD 3.2 WS
local 6bone Gateway connected to 6bone over gif tunnel
Dualstacked Firewall

For IPv4 NAT/PAT will be used, for IPv6 routable addresses will be used, which is part of the real challenge of this setup.
The link to the Internet is 2 MBit/s and will be used for tunneled as well as NAT/PAT traffic incoming and outgoing through the firewall.
The proposed firewall should be able to handle at least 50 clients without noteably slowing down traffic running through it.

### 1.5 Why OpenBSD as firewall OS ?

OpenBSD is well known to be a security aware operating system. It can be installed on disks with 100MB or even smaller without loosing much of its functionality. The system has IPv6 support since version 2.7 and the IPv6 is known to be solid. Also the included packet filter PF since version 3.0 (earlier versions had Darren Reeds IPF included) has a very good reputation among experts and the rules are very easy to write and read and support the definition of macros and other very valuable features.

# 2 Setting up the OpenBSD base system

## 2.1 Choosing the hardware

Our aim is to have a firewall that is able to handle a 2MBit/s synchronous link without slowing down communication significantly. Also we have to keep in mind that NAT and other stateful inspection techniques might increase the load on the machine compared to simply forwarded packets. Our ruleset will only partly be performance improved and we need a slightly faster CPU for this too.
The testing will be done with a Intel Pentium© 180MHz with 140MB RAM and two 10MBit/s PCI

network cards. The available storage is a 1600MB IDE harddisk, this could even be done with (for example) a 250MB disk or less with a stripped down version of OpenBSD. For 50 machines behind your firewall, I would recommend at least 64MB or better 128MB RAM for the state tables. Also you should use 100MBit/s cards if possible but in any case only use PCI cards and not any ISA cards because of their faster throughput on the PCI bus.
Make sure that the hardware you are planning to use is listed on the “OpenBSD supported hardware list”[4].

## 2.2 Installing OpenBSD

We will use the latest released version of OpenBSD for this setup, which is at the time of this writing 3.2. To install the base system we need to create a bootable medium, either 1.44MB floppy disk or a CDROM disk. Installation can be done from an attached medium like disk (i.e. harddisk, CDROM etc.) or over network. We will install using a CDROM and install all software from an anonymous FTP server.

Now we follow the instructions from the OpenBSD installation page to setup a working base system. The OpenBSD FAQ chapter 4[5] contains detailed instructions on how to create boot disks and initially setting up a base system. Depending on what size of harddisk you use you can increase the disklabel partioning recommendations (see table below) as it fits your needs. The values below just represent a moderate minimum necessary to get the described setup working:

| *Partition* | *Size in Megabytes* | *Mount Point* |
|---|---|---|
| a | 32 | / |
| b | 64 | swap |
| d | 32 | /tmp |
| e | 128 | /var |
| g | 150 | /usr |
| h | 100 | /home |

At this point you should have decided on your internal and external IP adresses.

Only install the following three packages from the installation menu, after partitioning your disk and setting up your network:

- base3X.tgz
- etc3X.tgz
- bsd

If you want manual pages on your system, you should also select

- man3X.tgz

If you also plan to compile software on the system, which is not a recommended practice for a firewall system, you would also need

4 OpenBSD supported hardware list - http://www.openbsd.org/i386.html#hardware
5 The OpenBSD FAQ chapter 4 - http://www.openbsd.org/faq/faq4.html .

- misc3X.tgz
- comp3X.tgz

Keep in mind that although you will make the attackers work a bit harder if there are no compilers installed, this will not keep them or her from installing the necessary files themselves after successfully intruding the site.

For all further configurations described herein, the first 3 selected packages will fully suffice.

Answer “No” when asked if you want to run X on your machine, you definitely don't want to.

Set your timezone accordingly, in our case to “Europe/Zurich” and reboot your system.

### 2.2.1 Tightening and tweaking the OpenBSD system

#### *2.2.1.a Disable unused services*

First of all, we disable all unused services on the freshly installed firewall system. The following services are the only ones uncommented in the /etc/inetd.conf:

- identd
- daytime
- time
- comsat

Not that any of those services has any known security problems, but one should always run the least services needed to do the given job. Since we want to disable all those services, it wil be easier to disable the inetd daemon per se, by changing in /etc/rc.conf the line

```
inetd=YES              # almost always needed
```

to

```
inetd=NO              # almost always needed
```

Don't let the comment puzzle you, while this may be true for different sort of network hosts, it is not needed and not advisable for a firewall, we will deal with identd queries later. Every service running is one more software package that you have to monitor for vulnerabilities and fix them immediately in case they arise, set aside that our firewall has a far better answer to identd queries.

To activate immediately the changes made, either reboot or kill the inetd process with a command like:

```
kill `cat /var/run/inetd.pid `
```

You should make sure that the inetd daemon really stopped by grepping the output of the ps command:

```
ps auxwww | grep inetd
```

But you can also just leave it as it is, we are going to reboot the system later anyway.

#### *2.2.1.b Optimize disk performance and consistency*

To reduce the chances of loosing or corrupting data after a power loss or system crash we are going to enable soft dependencies. This results basically in metadata not being written immediately but ordered in a way so that it reflects the files actually on disk. The command man 8 mount gives you more information if you have the man pages installed, in our case you can read the man page online[6]. To enable soft dependencies we need the following options in /etc/fstab to our mount points:

```
/dev/wd0a / ffs rw,softdep 1 1
/dev/wd0h /home ffs rw,nodev,nosuid,softdep 1 2
/dev/wd0d /tmp ffs rw,nodev,nosuid,softdep 1 2
/dev/wd0g /usr ffs rw,nodev,softdep 1 2
/dev/wd0e /var ffs rw,nodev,nosuid,softdep 1 2
```

(Note the added keyword softdep at the end of the mount options)

#### 2.2.1.c Ways to further lock down the firewall

Although OpenBSD has proved to be highly secure in its default configuration, there are few things that can be done additionally to further lock down the firewall system against attackers.

The following list should only serve as a starting point for somebody looking to further secure his system since a detailed explanation of each issue would be beyond the scope of this paper.

- Tightening rules even more
- Extensive use of sudo to avoid root logins at almost any price, see man sudo for more information
- Tightening OpenSSH configuration, see man sshd_config for more information
- Using systrace to limit what programs may do on your system, see man systrace for more information

## 2.3 Getting IPv6 address space

While this is not topic of this paper, the following steps might help you to get started:

1. Your first stop should be the 6bone hookup document which introduces you to the process of requesting and setting up IPv6 address space from a tunnel broker or even natively. http://www.6bone.net/6bone_hookup.html
2. Find a tunnel broker in your country or at least on your continent (this might sound like a joke, but there are actually countries and continents with none or only very few IPv6 connections).

## 2.4 Preparing the (test-)clients

We have two test-clients, one running Windows 2000 SP2 and the IPv6 Hotfix[7], the other running OpenBSD 3.2 like the firewall but it has been setup with all installation packages selected and X-Windows was configured after the installation.

Both machines have IPv4 and IPv6 addresses configured. The assigned internal IP addresses are listed in the following table:

| *Machine Name* | *OS* | *IPv4 address* | *IPv6 address* | *comment* |
|---|---|---|---|---|
| 6boner | OpenBSD | 172.16.0.40 | fec0:2029:f001:128::40 | firewall external interface |

6 OpenBSD mount manual page - http://www.openbsd.org/cgi-bin/man.cgi?query=mount&sektion=8
7 Available from Microsoft Corp. at http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp

| *Machine Name* | *OS* | *IPv4 address* | *IPv6 address* | *comment* |
|---|---|---|---|---|
| 6boner | OpenBSD | 192.168.192.1 | fec0:2029:f001:184:192::1 | firewall internal interface |
| amy | Windows 2000 | 192.168.192.2 | fec0:2029:f001:184:192::2 | desktop client |
| shark | OpenBSD | 192.168.192.3 | fec0:2029:f001:184:192::3 | internal file and mail server |

# 3 Configuring the firewall

## 3.1 Configuration changes to OpenBSD

Some more changes need to be done to allow our firewall system actually act as such.

### 3.1.1 Enabling IPv4 and IPv6 routing

Since our firewall has to route packets, we also need to instruct the kernel to do so since it is disabled by default. We can enable routing on our firewall with the following command:

```
sysctl -w net.inet.ip.forwarding=1
sysctl -w net.inet6.ip6.forwarding=1
```

To make this happen after every reboot, we uncomment the two respecting lines in /etc/sysctl.conf to look as follows:

```
net.inet.ip.forwarding=1        # 1=Permit forwarding (routing) of packets
net.inet6.ip6.forwarding=1      # 1=Permit forwarding (routing) of packets
```

This file will be parsed by the boot process resp. by the script /etc/rc and the resulting lines will be piped through the sysctl utility.

### 3.1.2 Enabling the packet filter pf

Finally we have to enable OpenBSD's packet filter at boot time. Again, edit /etc/rc.conf and change the line

```
pf=no                # Packet filter / NAT
```

to

```
pf=YES               # Packet filter / NAT
```

You don't need to reboot, you can enable your pf on the fly with the command:

```
pfctl -f /etc/pf.conf -e
```

## 3.2 The policy

We will first install a basic IPv4 ruleset and enable Networt/Port Address Translation (NAT/PAT) to allow our two clients to access the IPv4 part of the internet (and to download servicepacks and hotfixes needed i.e. for the Windows machine).

### 3.2.1 Internal to external

Our users shall be allowed to

- surf the web to servers on port 80 and 443 (SSL)
- download their mail from POP3, IMAP and IMAPS (SSL) servers, by allowing traffic to ports 110, 143 and 993
- send their mail out to the smtp server of our ISP
- query the two DNS servers of our ISP
- ping external hosts

### 3.2.2 Internal to firewall

Our users shall be allowed to

- login by secure shell for maintenance from selected admin machines
- ping the firewall for diagnostics

In the chapter “More advanced rules” we will add further permissions to our rules but for the moment this all that is allowed from the internal network to the Internet.

### 3.2.3 External to internal

External machines shall be restricted to do

- nothing

### 3.2.4 Firewall to external

The firewall itself shall be able to

- query the two DNS servers of our ISP
- mail reports to our POP3 / IMAP server

Also, in the above mentioned chapter we will introduce some controlled access to internal sites from the Internet. For the moment we don't allow any incoming connections at all.

### 3.2.5 Firewall to internal

The firewall itself doesn't need to talk to the internal network. Therefor the policy is to allow

- nothing

### 3.2.6 External to firewall

Nobody from outside shall be allowed to talk to our firewall from an external network. Therefore the policy is to allow

- nothing

## 3.3 IPv4 basic rule set

Our very basic policy results in the following basic ruleset in /etc/pf.conf[8]:

```
scrub in all fragment reassemble
block in log all
block return-rst in log on xl0 proto tcp from any to any port = auth
pass out on xl0 inet proto udp from 192.168.192.0/24 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 192.168.192.0/24 to 172.16.1.6 port = domain keep state
pass out on xl0 inet proto udp from 127.0.0.1 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 127.0.0.1 to 172.16.1.6 port = domain keep state
pass out on xl0 inet proto udp from 172.16.0.40 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 172.16.0.40 to 172.16.1.6 port = domain keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = smtp keep state
pass out on xl0 inet proto tcp from 127.0.0.1 to any port = smtp keep state
pass out on xl0 inet proto tcp from 172.16.0.40 to any port = smtp keep state
pass out on xl0 inet proto icmp all icmp-type echoreq code 0 keep state
pass in log on xl1 inet proto tcp from 192.168.192.0/24 to 192.168.192.1 port = ssh keep state
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = smtp
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = imaps
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = imap
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = pop3
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = https
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = www
pass in on xl1 inet proto icmp all icmp-type echoreq code 0
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = smtp keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = imaps keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = imap keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = pop3 keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = https keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = www keep state
pass in on xl1 inet proto udp from 192.168.192.0/24 to 172.16.0.3 port = domain
pass in on xl1 inet proto udp from 192.168.192.0/24 to 172.16.1.6 port = domain
pass in on xl1 inet proto tcp from 192.168.192.10 to 192.168.192.1 port = ssh
pass in on xl1 inet proto tcp from 192.168.192.11 to 192.168.192.1 port = ssh
```

A quite lengthy ruleset for what we're doing, so there is some room for streamlining.

The following ruleset is absolutely identical to the previous, but it uses some of pf's tricks to get better readable and shorter firewall rule files:

```
extif = "xl0"
intif = "xl1"
extip4 = "172.16.0.40"
intip4 = "192.168.192.1"
intnet4 = "192.168.192.0/24"
ispdns = "{ 172.16.1.6, 172.16.0.3 }"
admin_machines4 = "{ 192.168.192.10, 192.168.192.11 }"
scrub in all
nat on xl0 from 192.168.192.0/24 to any -> 172.16.0.40
block in log all
block return-rst in log on $extif inet proto tcp from any to any port = 113
pass out on $extif inet proto udp from { $extip4, 127.0.0.1, $intnet4 } to $ispdns port = 53 keep state
pass out on $extif inet proto tcp from { $extip4, 127.0.0.1, $intnet4 } to any port = 25 keep state
pass out on $extif inet proto icmp all icmp-type 8 code 0 keep state
pass in log on $intif inet proto tcp from $intnet4 to $intip4 port = 22 keep state
pass in on $intif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 }
```

8 OpenBSD pf.conf manual page - http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf&apropos=0&sektion=0

```
pass in on $intif inet proto icmp all icmp-type 8 code 0
pass out on $extif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass in on $intif inet proto udp from $intnet4 to $ispdns port = 53
pass in on $intif inet proto tcp from $admin_machines4 to $intip4 port = 22
```

It should be noted that since OpenBSD version 3.2 NAT and firewall configuration are in the same file /etc/pf.conf, previously NAT configuration was in the file /etc/nat.conf, so if you use an older version you have to split the configuration into two separate files.

### 3.3.1 Some more explanations

#### *3.3.1.a Macros*

We used macros as easy shortcuts and to eliminate typing errors. We also used brackets to collapse several rules into one line.

#### *3.3.1.b Keep state*

Keeping states not only simplifies the writing and maintenance of your rules, it also speeds up the firewall rule lookup process. Before going through the rule file to find matching rules to apply to a packet, the state table is consulted, if the packet matches an existing state the rule file is not further processed.

#### *3.3.1.c Block return-rst*

Also note that we are dropping all blocked packets silently except those to port 113 tcp which is the identd service. A lot of server applications try to contact the identd daemon on the connecting IP address to query a username and wait for an answer before they proceed. If we just drop incoming connections to identd we will have very slow login times with for example POP3 or IMAP. This is why we send a TCP packet with the RST flag set back to let the daemon know that it is not worth to wait for a reply from us and to proceed with the connection.

### 3.3.2 Extending the basic IPv4 rule set

There are a few more rules we want to add to make our firewall even more secure than it already is. This list should just serve as a starting point for further improvements of your firewall rules:

#### *3.3.2.a Flag filtering*

As described above, only the first packet in a keep state rule gets really evaluated against the whole rule base, all subsequent packets get matched against the state table. Therefore we could be a bit more particular in what kind of packets we allow to create a state. So we could extend our rule allowing outgoing connections from:

```
pass out on $extif proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } keep state
```

to:

```
pass out on $extif proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } flags S/SA keep state
```

#### *3.3.2.b State Modulation*

If you are running TCP/IP stack with a weak ISN you might want to “modulate” their connections state additionally. ISN means Initial Sequence Number and refers to the randomness with which a TCP/IP stack choses the sequence numbers it labels the packets to reference replies from the other side. The underlying problems are well described in the CERT Advisory CA-2001-09[9]

Further enhancing the above rule this gives us:

```
pass out on $extif proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } flags S/SA modulate state
```

which does further randomize the ISNs on all states created by this rule.

#### *3.3.2.c Antispoofing*

OpenBSD's pf from version 3.2 and higher has another very handy configuration option called antispoof. Calling antispoof for an interface automatically inserts rules to avoid packets with a source address from the respective network interface not to enter through any other interface.
Adding the following line after our NAT rule:

```
nat on xl0 from 192.168.192.0/24 to any -> 172.16.0.40

# antispoof
antispoof for lo0
antispoof for xl0 inet
antispoof for xl1 inet
```

results in the following rules being expanded at the top of your current rule base:

```
block in on ! lo0 inet from 127.0.0.1/8 to any
block in on ! lo0 inet6 from ::1 to any
block in on ! xl0 inet from 172.16.0.40/24 to any
block in inet from 172.16.0.40 to any
block in on ! xl1 inet from 192.168.192.1/24 to any
block in inet from 192.168.192.1 to any
```

So far for special tricks, for our dualstacked firewall we'll stay with our minimum ruleset to keep things simple and manageable.

### 3.3.3 Assigning IPv6 addresses

We will now save our basic IPv4 ruleset for later and start over with a new one for IPv6. Just copy the whole pf.conf away to a safe place and open a new one in /etc/pf.conf .

First of all you should have added the respective IPv6 addresses to the interfaces either by using a command like

```
ifconfig xl0 inet6 fec0:2029:f001:128::40 prefixlen 64 alias
```

or by adding a line like this to (in our example) /etc/hostname.xl0

```
inet6 alias fec0:2029:f001:128::40 64
```

We also need to add a default route:

```
route add -inet6 default fec0:2029:f001:128::1
```

To set this default route after reboot, you can add the above line at the end of your /etc/rc.local startup file.

9 CERT Advisory CA-2001-09 - http://www.cert.org/advisories/CA-2001-09.html

On the Windows 2000 machine you would enter something like this to enable a static IPv6 address on your network interface:

```
ipv6 adu 4/fec0:2029:f001:192::2
```

and a default route

```
ipv6 rtu ::/0 4/fec0:2029:f001:192::1
```

We will talk about autoconfiguration and the necessary changes to our ruleset to make it work later.

### 3.4 IPv6 basic ruleset

Naturally, our very same policy applies to IPv6 world as well, therefore we will now create the equivalent of the previous ruleset for IPv6:

```
extif = "xl0"
intif = "xl1"
extip6 = "fec0:2029:f001:128::20"
intip6 = "fec0:2029:f001:192::1"
intnet6 = "fec0:2029:f001:192::/64"
ispdns6 = "{ fec0:2029:f001:1::1, fec0:2029:f001:128::3 }"
admin_machines6 = “{ fec0:2029:f001:192::10, fec0:2029:f001:192::11 }”
antispoof for lo0
antispoof for xl0 inet
antispoof for xl1 inet
block in log all
block return-rst in log on $extif inet6 proto tcp from any to any port = 113
pass out on $extif inet6 proto udp from { $extip6, ::1, $intnet6 } to $ispdns6 port = 53 keep state
pass out on $extif inet6 proto tcp from { $extip6, ::1, $intnet6 } to any port = 25 keep state
pass out on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 136 } keep state
pass in on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 134, 135, 136 }
pass in log on $intif inet6 proto tcp from $intnet6 to $intip6 port = 22 keep state
pass in on $intif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 }
pass out on $extif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass in on $intif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 129, 135, 136 }
pass in on $intif inet6 proto udp from $intnet6 to $ispdns6 port = 53
pass in on $intif inet6 proto tcp from $admin_machines6 to $intip6 port = 22
```

which expands to the following ruleset:

```
block in on ! lo0 inet from 127.0.0.1/8 to any
block in on ! lo0 inet6 from ::1 to any
block in on ! xl0 inet from 172.16.0.40/24 to any
block in inet from 172.16.0.40 to any
block in on ! xl1 inet from 192.168.192.1/24 to any
block in inet from 192.168.192.1 to any
block in log all
block return-rst in log on xl0 inet6 proto tcp from any to any port = auth
pass out on xl0 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet6 proto udp from ::1 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from ::1 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:128::20 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:128::20 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from ::1 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:128::20 to any port = smtp keep state
pass out on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv keep state
pass out on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type echoreq keep state
pass in on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv
pass in on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbrsol
pass in log on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to fec0:2029:f001:192::1 port = ssh keep state
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imaps
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imap
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = pop3
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = https
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = www
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imaps keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imap keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = pop3 keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = https keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = www keep state
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type neighbrsol
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type echorep
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type echoreq
pass in on xl1 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:128::3 port = domain
pass in on xl1 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:184a::6 port = domain
pass in on xl0 inet6 proto tcp from fec0:2029:f001:192::10 to fec0:2029:f001:192::1 port = ssh
pass in on xl0 inet6 proto tcp from fec0:2029:f001:192::11 to fec0:2029:f001:192::1 port = ssh
```

As you can see, there are quite some differences between the IPv4 basic ruleset concerning the handling of ICMP messages.
In IPv4 there is no such thing as the ARP[10]-protocol in IPv4 which is used to query the Ethernet-address of a machine that we want to communicate with over IPv4.
With IPv6 this is different; IPv6 uses a set of ICMP messages to query the Ethernet-address of a host on the same link. This is accomplished by the pair:

- Neighbor Solicitation Message (icmp-type 135)[11]
- Neighbor Advertisement Message (icmp-type 136)[12]

While the first does the job of an arp query, the second are replies to these queries.

The IPv6 icmp types 128[13] and 129[14] are echo request and echo reply respectively and needed for ping to work. If you want to traceroute from the inside, you need to make sure that your traceroute uses icmp. Under Windows this is the default, under OpenBSD you can force traceroute6 to use icmp with the -I switch.

Also note that the scrub directive has been removed, since it doesn't act on IPv6 packets yet, it only normalizes IPv4 packets for the moment.

### 3.4.1 DNS issues

Please note that you can't use OpenBSD built in nslookup command to make queries over IPv6. This nslookup command can't speak IPv6. Anyway, all IPv6 aware applications can query IPv6 related records over the IPv6 protocols.

While our OpenBSD client is able to resolve names and IPs over IPv6 solely, Windows 2000 can not.

10 RFC1042 - A Standard for the Transmission of IP Datagrams over IEEE 802 Networks
11 RFC2461 - Neighbor Discovery for IP Version 6 (IPv6)
12 RFC2461 - Neighbor Discovery for IP Version 6 (IPv6)
13 RFC2463 - Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification
14 RFC2463 - Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification

Therefore the Windows 2000 client is still relying on a DNS server reachable over IPv4 for name and IP resolution.

So while an URL like http://[fec0:501:4819:2000:210:f3ff:fe03:4d0]/ does work in our setup to surf www.kame.net, it will not work with http://www.kame.net without IPv4 DNS being accessible. This will be solved by combining the rulesets but you should be aware of this fact when following a step by step procedure like in this paper and are not able to see any sites. The same is naturally valid for tools like tracert6.exe, ping6.exe, etc.

### 3.4.2 Browser issues

It should be noted that on Windows platforms the only choice for IPv6 enhanced browsing experience is Internet Explorer, while on Linux and BSD platforms Mozilla works over IPv6.

The hs247.com homepage lists many pointers to IPv6 capable applications[15].

## *3.5 Combining both rulesets*

Provided that both our rulesets have worked so far as expected, we will now try to combine both rulesets into one. I have tried to keep it as simple a possible and to group rules together as much as possible:

1. Add missing macros from the IPv4 basic ruleset
2. Add “scrub in all” directive
3. Add NAT (only applies to IPv4 connections)
4. Add all existing IPv6 rules a line below with “inet6” replaced by “inet”
5. Exchange references to IPv6 specific macros with the corresponding IPv4 macro
6. Manually adjust icmp handling

This procedure leaves us with the following ruleset:

```
extif = "xl0"
intif = "xl1"
extip4 = "172.16.0.40"
extip6 = "fec0:2029:f001:128::20"
intip4 = "192.168.192.1"
intip6 = "fec0:2029:f001:192::1"
intnet4 = "192.168.192.0/24"
intnet6 = "fec0:2029:f001:192::/64"
ispdns = "{ 172.16.1.6, 172.16.0.3 }"
ispdns6 = "{ fec0:2029:f001:184a::6, fec0:2029:f001:128::3 }"
admin_machines4 = “{ 192.168.192.10, 192.168.192.11 }”
admin_machines6 = “{ fec0:2029:f001:192::10, fec0:2029:f001:192::11 }”
scrub in all
nat on $extif inet from $intnet4 to any -> $extip4
antispoof for lo0
antispoof for xl0 inet
antispoof for xl1 inet
block in log all
block return-rst in log on $extif inet6 proto tcp from any to any port = 113
block return-rst in log on $extif inet proto tcp from any to any port = 113
pass out on $extif inet6 proto udp from { $extip6, ::1, $intnet6 } to $ispdns6 port = 53 keep state
```

15 HS247 - IPv6 News & Links - http://www.hs247.com

```
pass out on $extif inet proto udp from { $extip4, 127.0.0.1, $intnet4 } to $ispdns port = 53 keep state
pass out on $extif inet6 proto tcp from { $extip6, ::1, $intnet6 } to any port = 25 keep state
pass out on $extif inet proto tcp from { $extip4, 127.0.0.1, $intnet4 } to any port = 25 keep state
pass out on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 136 } keep state
pass in on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 135, 136 }
pass out on $extif inet proto icmp all icmp-type 8 code 0 keep state
pass in log on $intif inet6 proto tcp from $intnet6 to $intip6 port = 22 keep state
pass in log on $intif inet proto tcp from $intnet4 to $intip4 port = 22 keep state
pass in on $intif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 }
pass in on $intif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 }
pass out on $extif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass out on $extif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass in on $intif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 129, 135, 136 }
pass in on $intif inet proto icmp all icmp-type 8 code 0
pass in on $intif inet6 proto udp from $intnet6 to $ispdns6 port = 53
pass in on $intif inet proto udp from $intnet4 to $ispdns port = 53
pass in on $intif inet proto tcp from $admin_machines4 to $intip4 port = 22
pass in on $intif inet6 proto tcp from $admin_machines6 to $intip6 port = 22
```

which results in the following expanded ruleset:

```
scrub in all fragment reassemble
block in on ! lo0 inet from 127.0.0.1/8 to any
block in on ! lo0 inet6 from ::1 to any
block in on ! xl0 inet from 172.16.0.40/24 to any
block in inet from 172.16.0.40 to any
block in on ! xl1 inet from 192.168.192.1/24 to any
block in inet from 192.168.192.1 to any
block in log all
block return-rst in log on xl0 inet6 proto tcp from any to any port = auth
block return-rst in log on xl0 inet proto tcp from any to any port = auth
pass out on xl0 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet6 proto udp from ::1 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from ::1 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:128::20 to fec0:2029:f001:128::3 port = domain keep state
pass out on xl0 inet6 proto udp from fec0:2029:f001:128::20 to fec0:2029:f001:184a::6 port = domain keep state
pass out on xl0 inet proto udp from 192.168.192.0/24 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 192.168.192.0/24 to 172.16.1.6 port = domain keep state
pass out on xl0 inet proto udp from 127.0.0.1 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 127.0.0.1 to 172.16.1.6 port = domain keep state
pass out on xl0 inet proto udp from 172.16.0.40 to 172.16.0.3 port = domain keep state
pass out on xl0 inet proto udp from 172.16.0.40 to 172.16.1.6 port = domain keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from ::1 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:128::20 to any port = smtp keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = smtp keep state
pass out on xl0 inet proto tcp from 127.0.0.1 to any port = smtp keep state
pass out on xl0 inet proto tcp from 172.16.0.40 to any port = smtp keep state
pass out on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv keep state
pass out on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type echoreq keep state
pass in on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv
pass in on xl0 inet6 proto ipv6-icmp all ipv6-icmp-type neighbrsol
pass out on xl0 inet proto icmp all icmp-type echoreq code 0 keep state
pass in log on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to fec0:2029:f001:192::1 port = ssh keep state
pass in log on xl1 inet proto tcp from 192.168.192.0/24 to 192.168.192.1 port = ssh keep state
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imaps
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imap
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = pop3
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = https
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = www
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = smtp
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = imaps
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = imap
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = pop3
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = https
pass in on xl1 inet proto tcp from 192.168.192.0/24 to any port = www
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = smtp keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imaps keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = imap keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = pop3 keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = https keep state
pass out on xl0 inet6 proto tcp from fec0:2029:f001:192::/64 to any port = www keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = smtp keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = imaps keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = imap keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = pop3 keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = https keep state
pass out on xl0 inet proto tcp from 192.168.192.0/24 to any port = www keep state
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type neighbradv
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type neighbrsol
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type echorep
pass in on xl1 inet6 proto ipv6-icmp all ipv6-icmp-type echoreq
pass in on xl1 inet proto icmp all icmp-type echoreq code 0
pass in on xl1 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:128::3 port = domain
pass in on xl1 inet6 proto udp from fec0:2029:f001:192::/64 to fec0:2029:f001:184a::6 port = domain
pass in on xl1 inet proto udp from 192.168.192.0/24 to 172.16.0.3 port = domain
pass in on xl1 inet proto udp from 192.168.192.0/24 to 172.16.1.6 port = domain
pass in on xl1 inet proto tcp from 192.168.192.10 to 192.168.192.1 port = ssh
pass in on x1 inet proto tcp from 192.168.192.11 to 192.168.192.1 port = ssh
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::10 to fec0:2029:f001:192::1 port = ssh
pass in on xl1 inet6 proto tcp from fec0:2029:f001:192::11 to fec0:2029:f001:192::1 port = ssh
```

Now, as we have all the rules in place that reflect our policy, we are about to test if our setup really does what we want it to do.

## 3.6 Testing the rulesets

We will test our ruleset by checking our policy's statements line by line by appropriate testing methods. Tools involved in testing are

- nmap[16] (a version which is IPv6 capable)
- netcat (nc) and tcpdump for further investigations

First of all we will try to scan an external host while listening on the external interface which packets will make it through and if our results are according to our policy.

### 3.6.1 Testing internal to firewall/external

#### 3.6.1.a IPv4 from internal

First of all, let us check if our firewall does not respond on any port except 113/tcp from an internal machine which is not in the $admin_machines macro of our ruleset with a command like

16 nmap portscanner - http://www.insecure.org/nmap/

```
nmap -P0 -sTU 192.168.192.1
```

which should result in something similar to

```
Interesting ports on 192.168.192.1:
(The 3066 ports scanned but not shown below are in state: filtered)
Port       State       Service
25/tcp     closed      smtp
80/tcp     closed      http
110/tcp    closed      pop-3
143/tcp    closed      imap2
443/tcp    closed      https
993/tcp    closed      imaps

Nmap run completed -- 1 IP address (1 host up) scanned in 282.431 seconds
```

To test from internal to external we will try to portscan our ISPs POP3 server and we will watch on our firewall's external interface which packets make it through.
We run tcpdump on the firewall and instruct it to listen only to our external interface and redirect its output to file for later analysis, with a command like

```
tcpdump -nvvv -i xl0 > /tmp/intext4.dmp
```

and on the internal portscanning machine we run a nmap portscan against a non-existing target, i.e.

```
nmap -P0 -sTU 172.16.1.14
```

When you display the contents of /tmp/intext4.dmp you should see a group of lines for each allowed port like the ones below for port 110/tcp but nothing more:

```
  16:31:47.982320 172.16.0.40.64848 > 172.16.0.120.110: S [tcp sum ok] 2940611853:2940611853(0) win 16384
<mss 14
  60,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 226066400 0> (DF) (ttl 63, id 47087, bad cksum 46e5!)
  16:31:47.982749 172.16.0.120.110 > 172.16.0.40.64848: S [tcp sum ok] 2695391048:2695391048(0) ack
2940611854 wi
  n 32120 <mss 1460,sackOK,timestamp 199337061 226066400,nop,wscale 0> (DF) (ttl 64, id 25742)
  16:31:47.983550 172.16.0.40.64848 > 172.16.0.120.110: . [tcp sum ok] ack 1 win 17376 <nop,nop,timestamp
2260664
  00 199337061> (DF) (ttl 63, id 57616, bad cksum 1dd0!)
  16:31:47.984709 172.16.0.40.64848 > 172.16.0.120.110: R [tcp sum ok] 1:1(0) ack 1 win 0 (DF) (ttl 63, id 47684,
   bad cksum 44a8!)
```

Your dump should not contain any line containing a destination port other than what we allowed in our ruleset.

### *3.6.1.b IPv6 from internal*

The IPv6 portscan of our firewall should give identical results to the scans done over IPv4 because our policy specifies them to behave completely identical. So we scan our firewall from internal over IPv6 with a command like

```
nmap -P0 -sT -6 fec0:2029:f001:192::1
```

and the result should look like

```
Interesting ports on fec0:2029:f001:192::1:
(The 1598 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
25/tcp     closed      smtp
80/tcp     closed      http
110/tcp    closed      pop-3
```

```
143/tcp    closed      imap2
443/tcp    closed      https
993/tcp    closed      imaps

Nmap run completed -- 1 IP address (1 host up) scanned in 185.266 seconds
```

### 3.6.2 Testing external to firewall/internal

Nmap will be started twice, first to scan the external single IPv4 address and then the external single IPv6 address as well as the complete internal IPv6 network.

#### *3.6.2.a IPv4 from external*

Issue the following command from a machine which is connected to the external network anywhere (read: the Internet) but without any filtering in place enroute:

```
nmap -P0 -sTU 172.16.0.40
```

The option -P0 is necessary since our firewall will not reply to icmp echo requests (ping), and nmap would stop portscanning when not given this option. The results should look similar to:

```
Interesting ports on 6boner.telemedia.ch (172.16.0.40):
(Ports scanned but not shown below are in state: filtered)
Port       State       Service
113/tcp    unfiltered  auth

Nmap run completed -- 1 IP address (1 host up) scanned in 414 seconds
```

(Remember the return-rst statement for port 113 in our rulesets)

#### *3.6.2.b IPv6 from external*

Now let's repeat this over IPv6 for the firewall's external IPv6 address, with the limitation that the nmap scanners latest alpha release (3.10ALPHA9) at the time of this writing only supports tcp connect scans, but no form of udp scanning. Therefore we leave off the -U option:

```
nmap -P0 -sT -6 fec0:2029:f001:128::40
```

which should result in a similar output like the following:

```
Interesting ports on fec0:2029:f001:128::40:
(The 1604 ports scanned but not shown below are in state: filtered)
Port       State       Service
113/tcp    closed      auth

Nmap run completed -- 1 IP address (1 host up) scanned in 498.963 seconds
```

Currently scanning on IPv6 with nmap is only available for individual hosts, but not for networks, therefore we will only scan our existing internal clients explicitly, by specifying the same command as above but with the clients IPv6 address as argument, i.e.:

```
nmap -P0 -sT -6 fec0:2029:f001:192::2
```

which will result in something like

```
Interesting ports on fec0:2029:f001:192::2:
(The 1604 ports scanned but not shown below are in state: filtered)
Port       State       Service
113/tcp    closed      auth

Nmap run completed -- 1 IP address (1 host up) scanned in 797.166 seconds
```

We can now assume that our firewall does what our policy wants it to do. In real life you would now start to further tighten your ruleset. Some hints have been given within this paper as where to look to improve the firewalls system security as well as the firewalls ruleset.

# 4 More advanced rules

## 4.1 Allowing the clients to autoconfigure their IPv6 addresses

There are two methods to autoconfigure the clients' IP addresses in IPv6:

- stateless address autoconfiguration
- stateful address autoconfiguration

Stateful address autoconfiguration actually refers to a DHCP-like implementation, where a server keeps track of assignments, leases etc. While with stateless address autoconfiguration the host in question generates its address on its own using local information like the MAC address of the interface concerned and site information which is advertised by the local router. Stateless address autoconfiguration is explained in RFC2462[17].

The author is not aware of a DHCP implementation for IPv6 at the moment, therefore we will show how to implement stateless address autoconfiguration. Also stateless autoconfiguration does not require any manual configuration on the client side which sounds like plug-n-play to us.

For stateless autoconfiguration to work we need a router that advertises our internal prefix to our clients, this will be our firewall. The daemon we want to run on our firewall is rtadvd[18]

So the steps are:

1. start the rtadvd daemon by invoking it with the internal interface as argument, in our case: *\/usr/sbin/rtadvd xl1*
2. add the following lines at the end of our /etc/pf.conf:

```
pass in on $intif inet6 proto ipv6-icmp all ipv6-icmp-type 133
pass out on $intif inet6 proto ipv6-icmp all ipv6-icmp-type 134
```

This is all that is required to let our clients autoconfigure from the firewall.

## 4.2 Allowing inbound traffic to a internal dualstacked mail- and webserver

At a later point, we do not want to fetch our mail anymore from our ISP by POP3 and IMAP. Rather we would like our ISP to let the mx-entry of our domain point to our firewalls external IPv4 and IPv6 address to receive our mail directly on our internal mail server.

Also our website has been improved to house database driven content management system and we need to run that on our internal server too. Now let's see how we can enhance our rules to allow connections from external to our internal server at 192.168.192.3 / fec0:2029:f001:192::3.

It should be mentioned, that the security of your internal mail-/webserver is now crucial to the security of the whole internal network. A much more secure approach would be to put the mail-/webserver into a separate network segment attached to the firewall, often called Demilitarized Zone

17 RFC2462 - IPv6 Stateless Address Autoconfiguration
18 OpenBSD rtadvd manual page - http://www.openbsd.org/cgi-bin/man.cgi?query=rtadvd&sektion=0

(DMZ) or Secure Server Network (SSN).

Given our ISP has changed all DNS entries accordingly, we can now make the necessary changes to our ruleset.

### 4.2.1 Inbound IPv4

We will first add all changes needed to access our internal server from the Internet over IPv4. Since with IPv4 we do not have routable addresses we first have to add a redirector statement to redirect incoming traffic coming on ports 25/tcp and 80/tcp to our internal mailserver. For this we add the following lines immediately after our nat statement in /etc/pf.conf:

```
rdr on $extif inet proto tcp from any to $extip4 port 25 -> $intsrv4 port 25
rdr on $extif inet proto tcp from any to $extip4 port 80 -> $intsrv4 port 80
```

Since we work with macros, we have to add the following macros at the end of the macro section in our /etc/pf.conf

```
intsrv4 = "192.168.192.3"
intsrv6 = "fec0:2029:f001:192::3"
```

Now our server is still not accessible, because incoming packets get blocked before our firewall has a chance to redirect them to the internal network. Therefore we need to explicitly allow them also on the external interface with the following pass rule added at the bottom of our /etc/pf.conf:

```
pass in on $extif inet proto tcp from any to $intsrv4 port = 25 keep state
pass in on $extif inet proto tcp from any to $intsrv4 port = 80 keep state
```

Our internal server is now reachable from the internet on ports 25/tcp and 80/tcp over IPv4.

### 4.2.2 Inbound IPv6

To allow IPv6 connections from the IPv6 Internet to our internal server we do not need to specify a redirector first, since we are running with routable ip addresses. Since we are blocking all initially, we need to do the work that rdr does for us in IPv4 manually, which means we have first to allow the packets to enter our firewall on the external interface. At this point the rdr statement creates a state for the internal interface so that we do not need to special allow our packets to leave the firewall on the internal interface. In IPv6 we need to explicitly allow both due to lack of NAT / RDR:

```
pass in on $extif inet6 proto tcp from any to $intsrv6 port = 25 keep state
pass in on $extif inet6 proto tcp from any to $intsrv6 port = 80 keep state
pass in on $intif inet6 proto tcp from $intsrv6 port = 25 to any keep state
pass in on $intif inet6 proto tcp from $intsrv6 port = 80 to any keep state
```

## *4.3 Complete /etc/pf.conf rule file*

For your reference here is the complete /etc/pf.conf with all changes added:

```
extif = "xl0"
intif = "xl1"
extip4 = "172.16.0.40"
extip6 = "fec0:2029:f001:128::20"
intip4 = "192.168.192.1"
intip6 = "fec0:2029:f001:192::1"
intnet4 = "192.168.192.0/24"
intnet6 = "fec0:2029:f001:192::/64"
ispdns = "{ 172.16.1.6, 172.16.0.3 }"
ispdns6 = "{ fec0:2029:f001:184a::6, fec0:2029:f001:128::3 }"
```

```
admin_machines4 = "{ 192.168.192.10, 192.168.192.11 }"
admin_machines6 = "{ fec0:2029:f001:192::10, fec0:2029:f001:192::11 }"
intsrv4 = "192.168.192.3"
intsrv6 = "fec0:2029:f001:192::3"
scrub in all
nat on $extif inet from $intnet4 to any -> $extip4
rdr on $extif inet proto tcp from any to $extip4 port 25 -> $intsrv4 port 25
rdr on $extif inet proto tcp from any to $extip4 port 80 -> $intsrv4 port 80
antispoof for lo0
antispoof for xl0 inet
antispoof for xl1 inet
block in log all
block return-rst in log on $extif inet6 proto tcp from any to any port = 113
block return-rst in log on $extif inet proto tcp from any to any port = 113
pass out on $extif inet6 proto udp from { $extip6, ::1, $intnet6 } to $ispdns6 port = 53 keep state
pass out on $extif inet proto udp from { $extip4, 127.0.0.1, $intnet4 } to $ispdns port = 53 keep state
pass out on $extif inet6 proto tcp from { $extip6, ::1, $intnet6 } to any port = 25 keep state
pass out on $extif inet proto tcp from { $extip4, 127.0.0.1, $intnet4 } to any port = 25 keep state
pass out on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 136 } keep state
pass in on $extif inet6 proto ipv6-icmp all ipv6-icmp-type { 135, 136 }
pass out on $extif inet proto icmp all icmp-type 8 code 0 keep state
pass in log on $intif inet6 proto tcp from $intnet6 to $intip6 port = 22 keep state
pass in log on $intif inet proto tcp from $intnet4 to $intip4 port = 22 keep state
pass in on $intif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 }
pass in on $intif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 }
pass out on $extif inet6 proto tcp from $intnet6 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass out on $extif inet proto tcp from $intnet4 to any port { 80, 443, 110, 143, 993, 25 } keep state
pass in on $intif inet6 proto ipv6-icmp all ipv6-icmp-type { 128, 129, 135, 136 }
pass in on $intif inet proto icmp all icmp-type 8 code 0
pass in on $intif inet6 proto udp from $intnet6 to $ispdns6 port = 53
pass in on $intif inet proto udp from $intnet4 to $ispdns port = 53
pass in on $intif inet proto tcp from $admin_machines4 to $intip4 port = 22
pass in on $intif inet6 proto tcp from $admin_machines6 to $intip6 port = 22
pass in on $intif inet6 proto ipv6-icmp all ipv6-icmp-type 133
pass out on $intif inet6 proto ipv6-icmp all ipv6-icmp-type 134
pass in on $extif inet proto tcp from any to $intsrv4 port = 25 keep state
pass in on $extif inet proto tcp from any to $intsrv4 port = 80 keep state
pass in on $extif inet6 proto tcp from any to $intsrv6 port = 25 keep state
pass in on $extif inet6 proto tcp from any to $intsrv6 port = 80 keep state
pass in on $intif inet6 proto tcp from $intsrv6 port = 25 to any keep state
pass in on $intif inet6 proto tcp from $intsrv6 port = 80 to any keep state
```

## 5 References

- Fuller, Vince; Li, Tony; Yu, Jessice, Varadhan, Kannan. “Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy”. RFC1519. September 1993.
  URL: http://www.ietf.org/rfc/rfc1519.txt (23 December 2002).
- Rekhter, Yakov; Moskowitz, Robert G; Karrenberg, Daniel; de Groot, Geert Jan,  Lear, Eliot. “Address Allocation for Private Internets”. RFC1918. February 1996.
  URL: http://www.ietf.org/rfc/rfc1918.txt (23 December 2002).
- Bradner, Scott; Mankin, Allison. “The Recommendation for the IP Next Generation Protocol”. RFC1752. January 1995.
  URL: http://www.ietf.org/rfc/rfc1752.txt (26 January 2003).
- Postel, Jon; Reynolds, Joyce K. “A Standard for the Transmission of IP Datagrams over IEEE 802

Networks”. RFC1042. February 1988.
URL: http://www.ietf.org/rfc/rfc1042.txt (26 January 2003).

- Narten, Thomas; Nordmark, Erik; Simpson, William Allen. “Neighbor Discovery for IP Version 6 (IPv6)“. RFC2461. December 1998.
URL: http://www.ietf.org/rfc/rfc2461.txt (26 January 2003).
- Thomson, Susan; Narten, Thomas. “IPv6 Stateless Address Autoconfiguration”. RFC2462. December 1998.
URL: http://www.ietf.org/rfc/rfc2462.txt (26 January 2003).
- Conta, Alex; Deering, Stephen. “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification”. RFC2463. December 1998.
URL: http://www.ietf.org/rfc/rfc2463.txt (26 January 2003).
- Mickey. ”Supported Hardware”. OpenBSD/i386. v1.335. 24 January 1998.
URL: http://www.openbsd.org/i386.html#hardware (26 January 2003).
- Holland, Nick. “4 - OpenBSD 3.2 Installation Guide”.  OpenBSD Frequently Asked Questions. 1.119. 1 January 2003.
URL: http://www.openbsd.org/faq/faq4.html
- “OpenBSD manual pages”.
URL: http://www.openbsd.org/cgi-bin/man.cgi (27 January 2003)
- CERT/CC. “CERT Advisory CA-2001-09 Statistical Weaknesses in TCP/IP Initial Sequence Numbers”. CERT/CC Advisories. 13 September 2002.
URL: http://www.cert.org/advisories/CA-2001-09.html (27 January 2003).
- Microsoft Corporation. “ Microsoft IPv6 Technology Preview for Windows 2000”.
URL: http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp (26 January 2003).
- Microsoft Corporation. “Frequently Asked Questions about the Microsoft IPv6 Technology Preview for Windows 2000”.
URL: http://msdn.microsoft.com/downloads/sdks/platform/tpipv6/faq.asp (26 January 2003).
- Fyodor. “Nmap -- Free Stealth Port Scanner For Network Exploration & Security Audits”. Insecure.org. 10 August 2002.
URL: http://www.insecure.org/nmap/