Solaris and IP Filter: How To Make Them Your NAT Solution

Solaris and IP Filter: How to Make Them Your NAT Solution by Kristy Westphal
last updated May 16, 2001

IP Filter - Freeware to Protect Your Perimeter

So, you are one of those 'lucky' system administrators in a shop that just spent the last dime of your limited budget on a new file server, and you don't get any more money until the next calendar year. And you were just hit by a hacker from the outside because you don't have any perimeter protection. To make matters even worse, you have only one or two assigned public IP addresses to access the Internet.

Things are looking rather bleak right now.

Well, fear not. There is a freeware packet-filter program that runs on numerous versions of UNIX, such as Solaris, FreeBSD, Irix, and HP-UX, that can provide both firewall services and network address translation (NAT). It is feature-rich and flexible and, since it is freeware, the price is especially right for everything from a small home or business network to a larger network. It's called IP Filter and is based on software that came out of the University of California at Berkeley.

This article will examine the ways in which IP Filter can be used for Network Address Translation on a Solaris system. For the purposes of this discussion, the author will be using IPFilter on a Solaris 7, x86 platform. Since the scope of this article is limited to IP Filter's NAT capabilities, for more in-depth discussions of the firewall functionality of IP Filter, it is suggested that readers refer to the following articles:

- Introduction to IP Filter, Part Two by Jeremy Rauch, SecurityFocus;
- How to Secure Your Solaris Server by Jamie Wilson, UNIXInsider; and,
- IP Filter Based Firewalls HOWTO by Erik Fichtner and Brendan Conoboy.

NAT Functionality

NAT is a firewall functionality that offers systems administrators the best use of their public IP addresses by allowing them to either map a single public IP address to multiple private ones on their network or multiple public addresses to multiple private ones. Ipnatis a tool that comes with IP Filter and gives it NAT functionality.

The ipnat utility has several functions itself, including letting the user:

- read in new rules (using the flag -f);
- delete a set of rules (-C flag for the rule listing, -F flag for the active entries);
- list active mappings (-l flag); and,
- view the current statistics (-s flag) of traffic that have run through the NAT gateway.

Having the ipnat utility available helps to enhance the security of the NAT gateway by allowing for dynamic changes to occur without a system reboot or a service restart.

Installing IP Filter as a NAT

Installation of IP Filter is very straightforward! With most freeware programs, users can expect massive amounts of special drivers and libraries to download. This is not the case with the IP Filter: all users need to get started is Solaris' native `make` function and the IP Filter tarball.

For the NAT configuration, users will need to do perform an additional step to ensure that ipforwarding is enabled on their kernel. First, they should check to see if it is enabled by running the following command:

```
ndd  -get /dev/tcp ip_forwarding
```

If this returns a 0, the following steps should be taken. However, if the initial step returns a 1, this step can be omitted.

To change this value to a 1 permanently, the user should add ipforwarding to the kernel by running the command:

```
ndd  -set /dev/tcp ip_forwarding 1
```

Running ndd will only change it for the time being. To change it permanently, the /etc/defaultrouter file should be removed (if it currently exists) and replaced by creating the /etc/gateways file (which can remain empty). With this file in place, the /etc/init.d/inetsvc startup script will change the value to a 1 upon each boot-up[1].

Users can download the latest version of IP Filter from: http://coombs.anu.edu.au/ipfilter/. The user can then untar the file and run "make solaris" in the directory that he or she wishes to place untar in. Once this process is complete, it can be installed like any other package on Solaris by running "make package". This will run through the package install process, which should present a prompt at some point asking the user whether or not he or she wants to install it. The user should respond yes, and then, once it is installed, the system will have binaries in /usr/kernel/drv, /opt/ipf and a newly installed directory at /etc/opt/ipf.

Note that upon install, ipmon is automatically installed in the startup scripts (see /etc/rc2.d/S65ipfboot for more details).

Issues to Watch For

As with any program, there are certain things in IP Filter that users need to be aware of prior to installation. One prominent warning that is issued on the home page for IP Filter is to avoid using `gmake` and to use the native `make` for Solaris instead. Another note that is made in the install text files specifically for Solaris is that if the following error occurs during the compile:

```
  /usr/local/lib/gcc-lib/sparc-sun-solaris2.3/2.6.3/include/sys/user.h:48,
from /usr/include/sys/file.h:15,                  from ../ip_nat.c:15:
/usr/include/sys/psw.h:19: #error Kernel include of psw.h
```

Then just comment out the line in: /usr/local/lib/gcc-lib/sparc-sun-solaris2.3/2.6.3include/sys/user.h which includes psw.h.

This is useful to know prior to its (unexpected) appearance on the screen. Other than these two issues, the install is very straightforward.

Configuring NAT: What goes into the rules

The user must create a file called /etc/opt/ipf/ipnat.rules, in which the NAT rules will reside. There

are four types of rules that go into the config file for NAT, they include:

- **Map** - the most basic type of NAT, which allows mapping of one address or multiple addresses to another single address or multiple address set;
- **Rdr** - which allows packets to be redirected from an IP address and port pairing to another IP address and port pairing;
- **Bimap** - which allows for bi-directional translation between an internal and external IP address; and,
- **map-block** - which allows for static IP address translation determined by an algorithm that puts the translated addresses into the destination range.

There are also flags that are used to make up the rules. These flags include:

- **port** or **ports** - distinguishes single or multiple ports that you wish to translate or forward;
- "**auto**" - is also used in conjunction with the 'port' command to indicate a range of ports to ensure that access to these simultaneous ports is available, rather than specifying any port within a range. Ports can be addressed either by number or by name, as long as it is registered in the /etc/services file;
- "**/bits**" or "**/mask**" or "**netmask**" - any of these formats can be used to indicate which netmask is to be used ('bits' means the CIDR address notation, such as /32= single address, /24 a class C address, etc.);
- **portmap** - to specify either tcp or udp ports, either in single or in multiples, with a ':' to separate multiples like so: '10000:40000'; and,
- **rr** - an option to specifiy round-robin format when selecting an IP address or port from a specified range.

There are also the standard "**from**", "**to**" and "**any**" words to help round out the rules. IP addresses can be used either by name or by number, as long as it is specified in the /etc/hosts file.

Configuring NAT: A Simple Configuration

The first step in configuring NAT is to determine what the user really wants to do with the NAT functionality. Once the appropriate method (either single public IP address to multiple private addresses or multiple public IP addresses to multiple private IP addresses) has been selected, the user can begin configuration. The pre-install step of adding ip_forwarding to the kernel will already have been completed. A simple configuration for single public IP addresses to a single private address might look something like the one presented below [1].

```
 map   nei0 192.168.5.0/32 -> 0/32   proxy   port   ftp   ftp/tcp map   nei0
192.168.5.0/32 -> 0/32   portmap   tcp/udp   10000:40000 map   nei0
192.168.5.0/32 -> 0/32
```

Some items to note in this single-to-single address configuration include:

1. Notice that the first rule allows ftp traffic in to the internal address; however, NAT is supposed to only deal with IP addresses, not ports too! Well, IP Filter can technically be called a NPAT (network and port address translation) because it can perform both. This can be very handy when the user is configuring the rules [2].
2. The second rule allows network access to ports 10000 through 40000 on this single address.
3. The last rule simply allows traffic to pass to and from the Internet on this box.
4. Notice the "0/32" notation for the destination. This can be helpful for those who dial up to an ISP that uses DHCP to assign public addresses. Other users can substitute this notation with their permanent single address.

5. nei0 is the external interface on my machine, but users can substitute something like ppp0 for a point- to-point, dial-up internet connection.

So what would the configuration look like for the same set of rules, but for multiple addresses to multiple addresses?

```
 map   nei0 192.168.5.0/24 -> 150.51.52.0/24   proxy   port   ftp   ftp/tcp map
nei0 192.168.5.0/24 -> 150.51.52.0/24   portmap   tcp/udp 10000:40000 map
nei0 192.168.5.0/24 -> 150.51.52.0/24
```

This configuration simply makes multiple rules for different subnets utilize separate public addresses.

Once the ruleset has been satisfactorily configured, the user should save the ipnat.rules file and restart ipmon by running:

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

The user should then test the ruleset to make sure it works properly. One thing that should be ensured when troubleshooting any problems with this setup is that the default route of the NAT node is pointed to the IP Filter box to correctly obtain an address.

More Complex NAT Configurations

Some of the more advanced features of NAT include the IP address and port forwarding features. This functionality offers users more granularity in the way in which the gateway can be configured. This is also where the redirection function comes into play in many cases.

NAT and Port Forwarding

The following is a hypothetical situation. Say that a system administrator has a server in the DMZ that needs to have telnet open, but he or she knows that leaving telnet open on port 23 is easy for a hacker to exploit. So, the admin sets up a NAT rule to redirect traffic from an unexpected port on one network address to forward all traffic to the telnet port on the server he or she need to access:

```
 rdr nei0 150.51.52.10/32 port 33 -> 150.51.52.11/32 port 23
```

To protect from other hosts trying to access telnet on the second server, users can configure a /etc/hosts.equiv file to only accept telnet connections from that IP address. The user can send telnet requests to the first host and get where they need to go and still confuse those that enjoy port scanning on the Internet.

In another hypothetical situation, say that DNS lookups for an external web server point to an IP address of 150.51.52.10. But this server lives in a DMZ, and the sysadmin prefers not to have an IIS or Netscape server necessarily running there because those services can be easily compromised. So, instead, the admin runs the web server on a server behind the firewall on port 8100, which is also not a privileged port. To redirect traffic from 150.51.52.10 on port 80 to this new server and port, the admin could add the follow NAT rule to his or her gateway as well as opening up this port for access through the firewall:

```
 rdr nei0 150.51.52.10/32 port 80 -> 192.168.5.12/32 port 8100
```

Disadvantages of Using IP Filter For NAT

IP Filter is very feature rich, which can be good and bad. It is good because there are so many things that users can do with IP Filter, depending on what their security goal is and how their choose to configure their firewall. The downside is that IP Filter can be configured in such a way that it may not work exactly as users expect. The key to solving this issue is to test everything - not just the new rule that has just been added, but all the rules in conjunction with this new rule as well.

Rule behavior can be unpredictable. When a new rule is thrown in the mix it may turn off something that the user did not intend to turn off. For instance, say that an administrator is trying to redirect traffic for a certain service to another server. If this server is on the same network segment, then the admin has effectively set up a reflector rather than an address redirection rule, which won't work anyway. Consider the following:

```
rdr nei0 150.50.52.3/24 port 21 ->  150.50.52.5/24 port 21
```

If the packet tries to go out of the same interface that it comes in on, the system will get confused.

Another situation in which it is easy to misconfigure the NAT gateway is when a user adds a rule that may already exist in some fashion in the active mappings. If the already-active mappings are not flushed out(using ipnat -F), then the rule might be useless. Good use of procedure in this case will alleviate any confusion.

If possible, users should set up a test environment in which to test rulesets prior to implementation. If this is not possible, then the ipnat does have a "-v" option, which turns on verbose so that users can see more information about how the rules are behaving.

Conclusions

IP Filter is not only an excellent perimeter defense mechanism for networks, but also a great way for the security-minded to teach themselves firewalling and NAT concepts. The key to making it work is in the way it is configured. Testing individual rules when changes are made is a necessary part of building any firewall. Monitoring log files is also important to ensuring that the firewall and NAT gateway behave as expected. It may take time to configure these services the right way, but there are numerous resources available (especially the HOWTO doc) to assist in the process. It is always better to take the time to do these things properly the first time. By doing it quickly and assuming that it is right, rather than testing the configuration, users risk leaving holes in these services which may well be found and exploited.

Kristy Westphal is a versatile network administrator, skilled in troubleshooting and process analysis. She is knowledgeable in several flavors of UNIX and NT, as well as various aspects of information security and disaster recovery planning. She is currently employed by KPMG LLP in their Information Risk Management practice as a consultant.

Bibliography

[1] "Solaris PPP/Nat Howto" by Rachel Polanskis. [2] "IPFilter Based Firewall HOWTO" by Brendan Conoboy and Erik Fichtner
March 10, 2001.

Relevant Links

IPFilter- TCP/IP Packet Filtering package

Introduction to IP Filter
Jeremy Rauch, SecurityFocus

Introduction to IP Filter, Part Two
Jeremy Rauch, SecurityFocus

Introduction

So, you've got several computers on your home or business network, and you'd like to be able to access the Internet from all of them, probably via a cable (or DSL) modem. Basically you have three options:

1. You connect all your machines and your cable modem to a hub, set them all up as DHCP clients (see this page for how to do this on Solaris), and go for it.
2. You set up one of your machines to do NAT (Network Address Translation), hiding the rest behind a firewall using RFC 1918 compliant addresses on your network.
3. You use one of those Netgear routers, or someting similar (e.g., those from Linksys), as your firewall, and let it perform NAT for you.

The last option is very popular, and is better than nothing, but you can't beat having your own dedicated firewall machine. The first method, as well as being insecure, lacks a certain *je ne sais quoi*, so I'll show you how to set up NAT using Darren Reed's IP Filter. If you want to use the first or last methods, you're on your own!

Hardware

In my experiments, I could only get NAT to work reliably when I had two physical interfaces (i.e., using two virtual interfaces, say `hme0` and `hme0:1`, didn't work). I used `hme1` to connect directly to my cable modem, and `hme0` as the connection to the rest of my network via a 100 baseT switch. `hme1` is under DHCP control per these instructions, and `hme0` was set up the conventional way, with the hostname in `/etc/hostname.hme0`, and the corresponding IP address in `/etc/hosts`.

Installing IP Filter

By far the best way to get IP Filter is install Solaris 10, which comes with Solaris IP Filter (which is based on IP Filter). For previous versions of Solaris, the best way to get IP Filter is to compile a copy of the latest source code, which can be downloaded from the IP Filter home page. As an alternative, I have a compiled version of the package here. This is IP Filter version 3.3.11, compiled on a Sun SPARCstation 20, running Solaris 2.6. I've also used it on a SPARCstation 2 running Solaris 7, but it is provided here without any support (I currently use the Solaris 10 version of IP Filter on a Sun Netra T1 105). You should probably download a more recent binary from Marauding Pirates.

Configuring IP Filter on Solaris 10

Once you've successfully installed IP Filter, you need to configure it. First of all, you need to make sure that your NAT box will forward IP packets (it's possible this ability was disabled for security reasons). As root, run this command:

```
routeadm
```

If the "Current Configuration" column of the "IPv4 forwarding" row says "disabled", then you must enable it. You do this by running the following command (again, as root):

```
routeadm -u -e ipv4-forwarding
```

The `-e ipv4-forwarding` option causes IPv4 forwarding to be enabled, and the `-u` flag causes the change to be applied to the running system (in addition to changing the settings when the system is next rebooted).

When you're happy that IP forwarding is enabled, you need to set up your NAT rules. The file `/etc/ipf/ipnat.conf` contains the rules you want to use. This is the `ipnat.conf` file I use, bearing in mind that all of my machines have an IP address in the 192.168.0.1 to 192.168.0.254 range; you should change the addresses between "`hme1`" and the "->" to suit your needs (note also that I've specified `hme1`; put the name of your outbound interface here instead):

```
map hme1 192.168.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map hme1 192.168.0.0/24 -> 0/32 portmap tcp/udp auto
map hme1 192.168.0.0/24 -> 0/32
```

The `0/32` stuff is some magic to tell IP Filter to use the address currently assigned to the interface - very useful in DHCP client environments!

The order of the rules is important; don't change them unless you know what you're doing, otherwise things will break! The first rule allows FTP access from all of your hosts. The second maps the source port numbers to a high range (10000 to 40000 by default), and the third rule maps all other TCP traffic.

Once you've set up your NAT rules, you need to enable packet filtering for the interface type you're using. This is done by uncommenting the appropriate line(s) in `/etc/ipf/pfil.ap`:

```
#le     -1      0       pfil
#qe     -1      0       pfil
hme     -1      0       pfil
```

When you're happy with your configuration, start the IP filter services:

```
svcadm restart network/pfil
svcadm restart ipfilter
```

The interfaces that you enabled packet filtering on by editing `/etc/ipf/pfil.ap` must be replumbed before you can use them. Here's how to do it, assuming your machine is set up like mine:

```
ifconfig hme1 unplumb
ifconfig hme1 plumb dhcp start
```

Another, perhaps easier, way is to simply reboot your machine. Although it smells like a typical Windoze "admin" kind of way of doing this, it does have the advantage of testing that your modifications will survive a reboot.

Assuming all is well, your firewall should now correctly handle NAT, even after a reboot. Assuming this is the case, enjoy! If this page has been useful to you, please consider buying a copy of my book, Solaris Systems Programming.

Configuring IP Filter for Previous Versions of Solaris

If you're using a version of Solaris prior to Solaris 10, and assuming you have Solaris 10-capable hardware, I don't know why you **wouldn't** use Solaris 10, here is the older version of these instructions. But really, you should upgrade to Solaris 10!

First of all, you need to make sure that your NAT box will forward IP packets (it's possible this ability was disabled for security reasons). As root, run this command:

```
ndd -get /dev/tcp ip_forwarding
```

If the result is "1", you're all set. Zero means that IP forwarding is not enabled. To enable it, delete

the file `/etc/notrouter`, and possibly `/etc/defaultrouter` too. Create an empty `/etc/gateways` file, and IP forwarding will be enabled at the next reboot.

One caveat applies, though: if you're using NAT and DHCP on the same server (like I do), IP forwarding will not get enabled. So, I install this script as `/etc/init.d/ip_forwarding`, with a symbolic link to it from `/etc/rc2.d/S69ip_forwarding`. With this script in place, IP forwarding will be enabled even if you are using a DHCP client.

When you're happy that IP Filter is running, and IP forwarding is enabled, you need to set up your NAT rules. The file `/etc/opt/ipf/ipnat.conf` contains the rules you want to use. This is the `ipnat.conf` file I use, bearing in mind that all of my machines have an IP address in the 192.168.0.1 to 192.168.0.254 range; you should change the addresses between "`hme1`" and the "`->`" to suit your needs (note also that I've specified `hme1`; put the name of your outbound interface here instead):

```
map hme1 192.168.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map hme1 192.168.0.0/24 -> 0/32 portmap tcp/udp auto
map hme1 192.168.0.0/24 -> 0/32
```

The `0/32` stuff is some magic to tell IP Filter to use the address currently assigned to the interface - very useful in DHCP client environments!

The order of the rules is important; don't change them unless you know what you're doing, otherwise things will break! The first rule allows FTP access from all of your hosts. The second maps the source port numbers to a high range (10000 to 40000 by default), and the third rule maps all other TCP traffic.

Use `/etc/init.d/ipfboot stop` and `/etc/init.d/ipfboot start` to test your configuration, and when you're happy that all is working well, reboot. This will make sure that everything still works as expected, even after a reboot.

That's about it - enjoy! If this page has been useful to you, please consider buying a copy of my book, Solaris Systems Programming.