11.3. DNAT target

The **DNAT** target is used to do Destination Network Address Translation, which means that it is used to rewrite the `Destination IP` address of a packet. If a packet is matched, and this is the target of the rule, the packet, and all subsequent packets in the same stream will be translated, and then routed on to the correct device, host or network. This target can be extremely useful, for example,when you have a host running your web server inside a *LAN*, but no real IP to give it that will work on the Internet. You could then tell the firewall to forward all packets going to its own HTTP port, on to the real web server within the *LAN*. We may also specify a whole range of destination IP addresses, and the **DNAT** mechanism will choose the destination IP address at random for each stream. Hence, we will be able to deal with a kind of load balancing by doing this.

Note that the **DNAT** target is only available within the PREROUTING and OUTPUT chains in the nat table, and any of the chains called upon from any of those listed chains. Note that chains containing **DNAT** targets may not be used from any other chains, such as the POSTROUTING chain.

Table 11-2. DNAT target

Option	**--to-destination**
Example	**iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1-192.168.1.10**
Explanation	The **--to-destination** option tells the DNAT mechanism which Destination IP to set in the IP header, and where to send packets that are matched. The above example would send on all packets destined for IP address 15.45.23.67 to a range of *LAN* IP's, namely 192.168.1.1 through 10. Note, as described previously, that a single stream will always use the same host, and that each stream will randomly be given an IP address that it will always be Destined for, within that stream. We could also have specified only one IP address, in which case we would always be connected to the same host. Also note that we may add a port or port range to which the traffic would be redirected to. This is done by adding, for example, an :80 statement to the IP addresses to which we want to DNAT the packets. A rule could then look like **--to-destination 192.168.1.1:80** for example, or like **--to-destination 192.168.1.1:80-100** if we wanted to specify a port range. As you can see, the syntax is pretty much the same for the **DNAT** target, as for the **SNAT** target even though they do two totally different things. Do note that port specifications are only valid for rules that specify the TCP or UDP protocols with the **--protocol** option.

Since **DNAT** requires quite a lot of work to work properly, I have decided to add a larger explanation on how to work with it. Let's take a brief example on how things would be done normally. We want to publish our website via our Internet connection. We only have one IP address, and the HTTP server is located on our internal network. Our firewall has the external IP address **$INET_IP**, and our HTTP server has the internal IP address **$HTTP_IP** and finally the firewall has the internal IP address **$LAN_IP**. The first thing to do is to add the following simple rule to the PREROUTING chain in the nat table:

```
iptables -t nat -A PREROUTING --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

Now, all packets from the Internet going to port 80 on our firewall are redirected (or **DNAT**'ed) to our internal HTTP server. If you test this from the Internet, everything should work just perfect. So, what happens if you try connecting from a host on the same local network as the HTTP server? It will simply not work. This is a problem with routing really. We start out by dissecting what happens in a normal case. The external box has IP address **$EXT_BOX**, to maintain readability.

1. Packet leaves the connecting host going to **$INET_IP** and source **$EXT_BOX**.
2. Packet reaches the firewall.
3. Firewall **DNAT**'s the packet and runs the packet through all different chains etcetera.
4. Packet leaves the firewall and travels to the $HTTP_IP.
5. Packet reaches the HTTP server, and the HTTP box replies back through the firewall, if that is the box that the routing database has entered as the gateway for **$EXT_BOX**. Normally, this would be the default gateway of the HTTP server.
6. Firewall Un-**DNAT**'s the packet again, so the packet looks as if it was replied to from the firewall itself.
7. Reply packet travels as usual back to the client **$EXT_BOX**.

Now, we will consider what happens if the packet was instead generated by a client on the same network as the HTTP server itself. The client has the IP address **$LAN_BOX**, while the rest of the machines maintain the same settings.

1. Packet leaves **$LAN_BOX** to **$INET_IP**.
2. The packet reaches the firewall.
3. The packet gets **DNAT**'ed, and all other required actions are taken, however, the packet is not **SNAT**'ed, so the same source IP address is used on the packet.
4. The packet leaves the firewall and reaches the HTTP server.
5. The HTTP server tries to respond to the packet, and sees in the routing databases that the packet came from a local box on the same network, and hence tries to send the packet directly to the original source IP address (which now becomes the destination IP address).

6. The packet reaches the client, and the client gets confused since the return packet does not come from the host that it sent the original request to. Hence, the client drops the reply packet, and waits for the "real" reply.

The simple solution to this problem is to **SNAT** all packets entering the firewall and leaving for a host or IP that we know we do **DNAT** to. For example, consider the above rule. We SNAT the packets entering our firewall that are destined for **$HTTP_IP** port 80 so that they look as if they came from **$LAN_IP**. This will force the HTTP server to send the packets back to our firewall, which Un-**DNAT**'s the packets and sends them on to the client. The rule would look something like this:

```
iptables -t nat -A POSTROUTING -p tcp --dst $HTTP_IP --dport 80 -j SNAT \
--to-source $LAN_IP
```

Remember that the POSTROUTING chain is processed last of the chains, and hence the packet will already be **DNAT**'ed once it reaches that specific chain. This is the reason that we match the packets based on the internal address.

This last rule will seriously harm your logging, so it is really advisable not to use this method, but the whole example is still a valid one. What will happen is this, packet comes from the Internet, gets SNAT'ed and DNAT'ed, and finally hits the HTTP server (for example). The HTTP server now only sees the request as if it was coming from the firewall, and hence logs *all* requests from the internet as if they came from the firewall.

This can also have even more severe implications. Take an SMTP server on the LAN, that allows requests from the internal network, and you have your firewall set up to forward SMTP traffic to it. You have now effectively created an open relay SMTP server, with horrenduously bad logging!

One solution to this problem is to simply make the SNAT rule even more specific in the match part, and to only work on packets that come in from our LAN interface. In other words, add a **--src $LAN_IP_RANGE** to the whole command as well. This will make the rule only work on streams that come in from the LAN, and hence will not affect the Source IP, so the logs will look correct, except for streams coming from our LAN.

You will, in other words, be better off solving these problems by either setting up a separate DNS server for your LAN, or to actually set up a separate DMZ, the latter being preferred if you have the money.

You think this should be enough by now, and it really is, unless considering one final aspect to this whole scenario. What if the firewall itself tries to access the HTTP server, where will it go? As it looks now, it will unfortunately try to get to its own HTTP server, and not the server residing on **$HTTP_IP**. To get around this, we need to add a **DNAT** rule in the OUTPUT chain as well. Following the above example, this should look something like the following:

```
iptables -t nat -A OUTPUT --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

Adding this final rule should get everything up and running. All separate networks that do not sit on the same net as the HTTP server will run smoothly, all hosts on the same network as the HTTP server will be able to connect and finally, the firewall will be able to do proper connections as well. Now everything works and no problems should arise.

Everyone should realize that these rules only affect how the packet is DNAT'ed and SNAT'ed properly. In addition to these rules, you may also need extra rules in the filter table (FORWARD chain) to allow the packets to traverse through those chains as well. Don't forget that all packets have already gone through the PREROUTING chain, and should hence have their destination addresses rewritten already by DNAT.

Works under Linux kernel 2.3, 2.4, 2.5 and 2.6.

Prev
CLASSIFY target

Home
Up

Next
DROP target