Linux Router

[**Introduction:**](#_h32u4dbm705z) **1**

[Network address translation:](#_qeh9i8xf6ijg) 1

[IP masquerading:](#_ild7j2501o6f) 1

[**Configuring Second Card:**](#_8n2bd7nf6zt) **2**

[/etc/sysconfig/network-scripts/ifcfg-br0:](#_9fmcw84ij0zc) 2

[/etc/sysconfig/network-scripts/ifcfg-enp6s0:](#_5g2l9t8racw9) 2

[**Create NAT’ing iptables file:**](#_quld496wkj8l) **3**

[**Routing Modifications to /etc/sysctl.conf:**](#_s8wlet33o816) **4**

# Introduction:

The class server can be used as a **Linux Router**. It can the techniques of **Network** **Address Translation** and **IP masquerading** to hide the **private** IP address space behind a single **public** IP address. This gives us the ability to increase the security of the private machines, limit the network “noise” coming from outside devices and optimize network utilization.

## Network address translation:

**NAT** is a method of remapping one IP address space into another by modifying network address information in the IP header of packets while they are in transit across a traffic routing device. The technique was originally used as a shortcut to avoid the need to readdress every host when a network was moved. It has become a popular and essential tool in conserving global address space in the face of IPv4 address exhaustion. One Internet-routable IP address of a NAT gateway can be used for an entire private network.

## IP masquerading:

IP masquerading is a technique that hides an entire IP address space, usually consisting of private IP addresses, behind a single IP address in another, usually public address space. The address that has to be hidden is changed into a single (public) IP address as "new" source address of the outgoing IP packet so it appears as originating not from the hidden host but from the routing device itself. Because of the popularity of this technique to conserve IPv4 address space, the term *NAT* has become virtually synonymous with IP masquerading.

# Configuring Second Card:

To basic hardware configuration for this set up is two Ethernet cards, one public and one private. By adjusting the internal netmask we can have as many as 16 million private machines that only need one public IP to access the Internet. Since private IP addresses are not routed on the Internet, these machines can exist and never be know by anyone outside the network.

The first step is to install the second Ethernet card and configure it. You can use the following command to find all available Ethernet devices:

|  |
| --- |
| ipconfig -a |

WIth output:

|  |
| --- |
| br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  inet 172.16.9.180 netmask 255.255.240.0 broadcast 172.16.15.255  ether b0:6e:bf:60:e6:ab txqueuelen 1000 (Ethernet)  RX packets 87854 bytes 28791094 (27.4 MiB)  RX errors 0 dropped 2787 overruns 0 frame 0  TX packets 11358 bytes 1873937 (1.7 MiB)  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  **enp0s31f6**: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  ether b0:6e:bf:60:e6:ab txqueuelen 1000 (Ethernet)  RX packets 359037 bytes 407711505 (388.8 MiB)  RX errors 0 dropped 0 overruns 0 frame 0  TX packets 12041 bytes 1973490 (1.8 MiB)  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  device interrupt 20 memory 0x92f00000-92f20000  **enp5s0**: flags=4098<BROADCAST,MULTICAST> mtu 1500  ether 00:10:18:12:0e:95 txqueuelen 1000 (Ethernet)  RX packets 637 bytes 47098 (45.9 KiB)  RX errors 0 dropped 0 overruns 0 frame 0  TX packets 296 bytes 19552 (19.0 KiB)  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  device interrupt 19 |

Now just as we did when creating a [network bridge](https://docs.google.com/document/d/1jlYH5ruIxnG5T7E1hEXe9ngZP_n1kk8MrULbIwm68t4/edit#heading=h.8j9ou2thfrks), we will configure the card.

From the output of the command above, find the Ethernet device of the new card. Assuming that the device is **enp5s0**, create:

* /etc/sysconfig/network-scripts/ifcfg-br0
* /etc/sysconfig/network-scripts/ifcfg-enp6s0

## /etc/sysconfig/network-scripts/ifcfg-br1:

|  |
| --- |
| IPADDR=10.0.0.1  NETMASK=255.0.0.0  BOOTPROTO="static"  DEVICE="br1"  TYPE="Bridge"  ONBOOT=yes  IPV6INIT=no |

## /etc/sysconfig/network-scripts/ifcfg-enp5s0:

|  |
| --- |
| TYPE="Ethernet"  BOOTPROTO="static"  DEVICE="enp5s0"  ONBOOT="yes"  BRIDGE="br1" |

Now disable NetworkManager and restart the network interfaces:

|  |
| --- |
| systemctl stop NetworkManager  systemctl disable NetworkManager  systemctl restart network |

This result in the both the public and the private network interfaces being configured, as well as the two Ethernet bridge interfaces. The bridge interfaces allow us to create Virtual Machines on either the private, the public or both network interfaces.

# Create NAT’ing iptables file:

Make sure that firewalld is stopped and disabled, and that the iptables packages are installed:

|  |
| --- |
| systemctl stop firewalld  systemctl disable firealld  yum install iptables iptables-utils iptables-services |

Now we need to add the **NAT’ing** information to top of **/etc/sysconfig/iptables**. In the box below, **enpXXX** is the **external** **Ethernet** interface. These **rules tell** the **server** to send all **outbound IP** packets with the r**eturn IP address of the server**. In our case, this is **172.16.9.180.**

|  |
| --- |
| # Firewall configuration written by system-config-securitylevel  # Manual customization of this file is not recommended.  \*nat  :PREROUTING ACCEPT [0:0]  :POSTROUTING ACCEPT [0:0]  :OUTPUT ACCEPT [0:0]  -A POSTROUTING -o **enp0s31f6** -j MASQUERADE  COMMIT |

The complete iptables file should now look like:

|  |
| --- |
| # Firewall configuration written by system-config-securitylevel  # Manual customization of this file is not recommended.  \*nat  :PREROUTING ACCEPT [0:0]  :POSTROUTING ACCEPT [0:0]  :OUTPUT ACCEPT [0:0]  -A POSTROUTING -o **enp0s31f6** -j MASQUERADE  COMMIT  \*filter  :INPUT ACCEPT [0:0]  :FORWARD ACCEPT [0:0]  :OUTPUT ACCEPT [0:0]  :INPUT - [0:0]  -A INPUT -j INPUT  -A FORWARD -j INPUT  -A INPUT -i lo -j ACCEPT  -A INPUT -p icmp --icmp-type any -j ACCEPT  -A INPUT -p 50 -j ACCEPT  -A INPUT -p 51 -j ACCEPT  -A INPUT -m state --state NEW -s 172.16.9.0/255.255.255.0 -m tcp -p tcp -j ACCEPT  -A INPUT -m state --state NEW -s 17.16.9.0/255.255.255.0 -m udp -p udp -j ACCEPT  -A INPUT -m state --state NEW -s 10.0.0.0/255.255.0.0 -m tcp -p tcp -j ACCEPT  -A INPUT -m state --state NEW -s 10.0.0.0/255.255.0.0 -m udp -p udp -j ACCEPT  -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT  -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT  -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT  -A INPUT -j REJECT --reject-with icmp-host-prohibited  COMMIT |

# Routing Modifications to /etc/sysctl.conf:

To turn on routing, we need to make some modifications to /etc/sysctl.conf.

**sysctl** is used to **modify** **kernel parameters at runtime**. You can use sysctl to both read and write sysctl data.

|  |
| --- |
| sysctl.conf is a simple file containing sysctl values to be read in and set by sysctl. The syntax is simply as follows:  # comment  ; comment  token = value  Note that blank lines are ignored, and whitespace before and after a token  or value is ignored, although a value can contain whitespace within.  Lines which begin with a # or ; are considered comments and ignored. |

The required changes are shown below:

|  |
| --- |
| # Kernel sysctl configuration file for Red Hat Linux  #  # For binary values, 0 is disabled, 1 is enabled. See sysctl(8) and  # sysctl.conf(5) for more details.  # Controls IP packet forwarding  net.ipv4.ip\_forward = 1  # Controls source route verification  net.ipv4.conf.default.rp\_filter = 1  # Do not accept source routing  net.ipv4.conf.default.accept\_source\_route = 0  # Controls the use of TCP syncookies  net.ipv4.tcp\_syncookies = 1 |

You can now load these changes using:

|  |
| --- |
| sysctl -p |