Linux Firewall & NAT

Linux Networking

Serhii Zakharchenko

Agenda

- Firewall technology overview
- Linux Firewall overview
- Iptables configuration
- Linux NAT overview

Firewall technology overview

Firewalls

- Firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules.
- A firewall typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.
- Characteristics of Firewalls:
  - **Physical barrier** - a firewall does not allow any external traffic to enter a system or a network without its allowance
  - **Multi-purpose** - a firewall has many functions other than security purposes, it can act as a router, network address translator, etc.
  - **Single transit point** – as e rule it is the only transit point between networks
  - **Flexible security policies** - different local systems or networks need different security policies.

Main Firewall Types

- **Packet-filtering firewall** — this is usually a router that examines some of the contents of the packets (looks at layer 3 and sometimes layer 4 information) according to a set of pre-established rules;
- **Stateful firewall** —keeps track of the connection state: whether the connection is in initiation, data transfer, or termination state;
- **Application gateway firewall (proxy firewall)** — analyzes information at OSI model layers 3, 4, 5, 7; usually implemented in software.
- **Host-based firewall** — server or workstation with firewall software running on it.
- **Transparent firewall** — it does not modify IP headers, filters IP traffic between a pair of bridged interfaces.

Packet filtering basics

- Packet filtering controls access to a network by analyzing the incoming and outgoing packets and passing or halting them based on stated criteria.
- Packet-filtering devices work at OSI layer 3,4 and use rules to determine whether to allow or deny traffic.
- Packet filtering devices can examine some information from the packet header, in particular:

  - Source IP address

  - Destination IP address

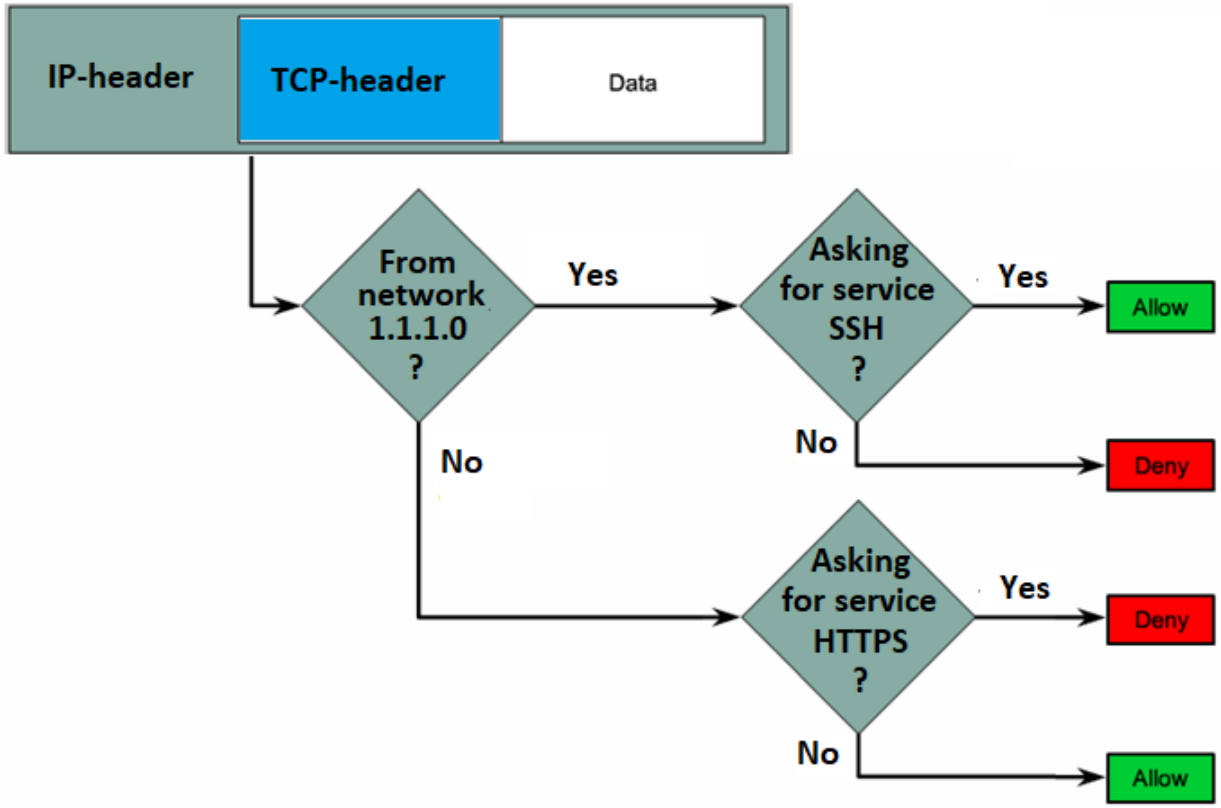
  - TCP/UDP source port

  - TCP/UDP destination port

  - ICMP message type

Packet filtering logic

- Each rule is aimed at analyzing a certain field (fields) of the packet header and contains an action - to allow the passage of the packet or to deny it.
- The rules are analyzed sequentially, step by step, from top to bottom.
- If a packet satisfied rule is found, the further rules review is terminated, and the appropriate action is performed with the package.



Rules analyzing sample

Packet filtering rule list:

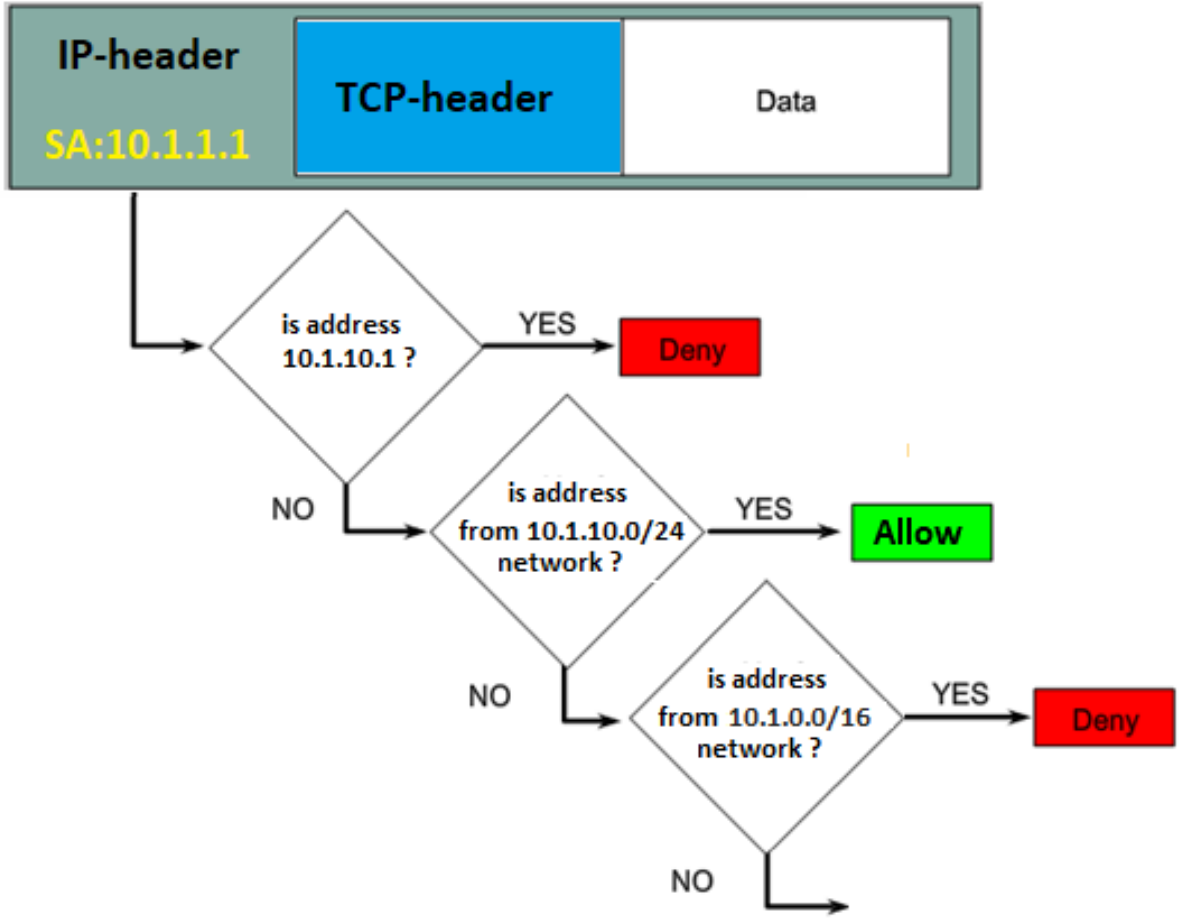
1 deny source IP address 10.1.10.1/32

2 allow source IP address 10.1.10.0/24

3 deny source IP address 10.1.0.0/16

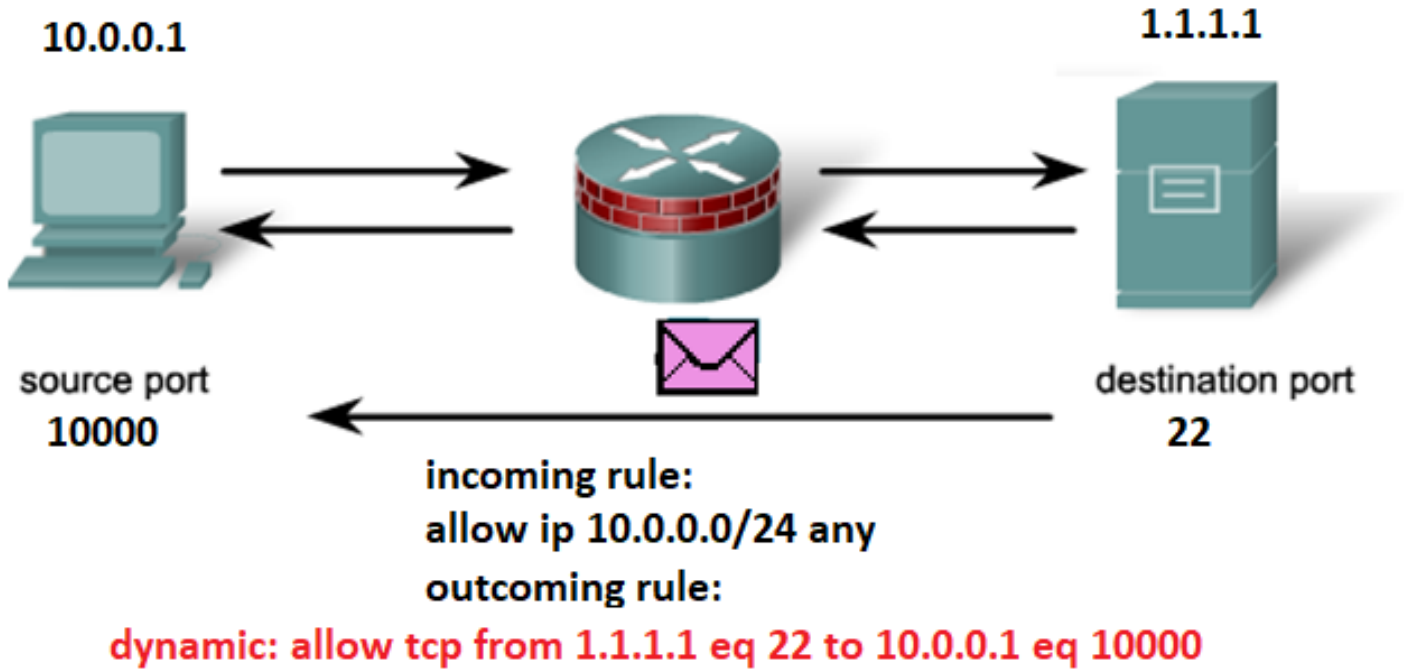
4 allow source IP address 10.0.0.0/8

5 deny source IP address 0.0.0.0/0



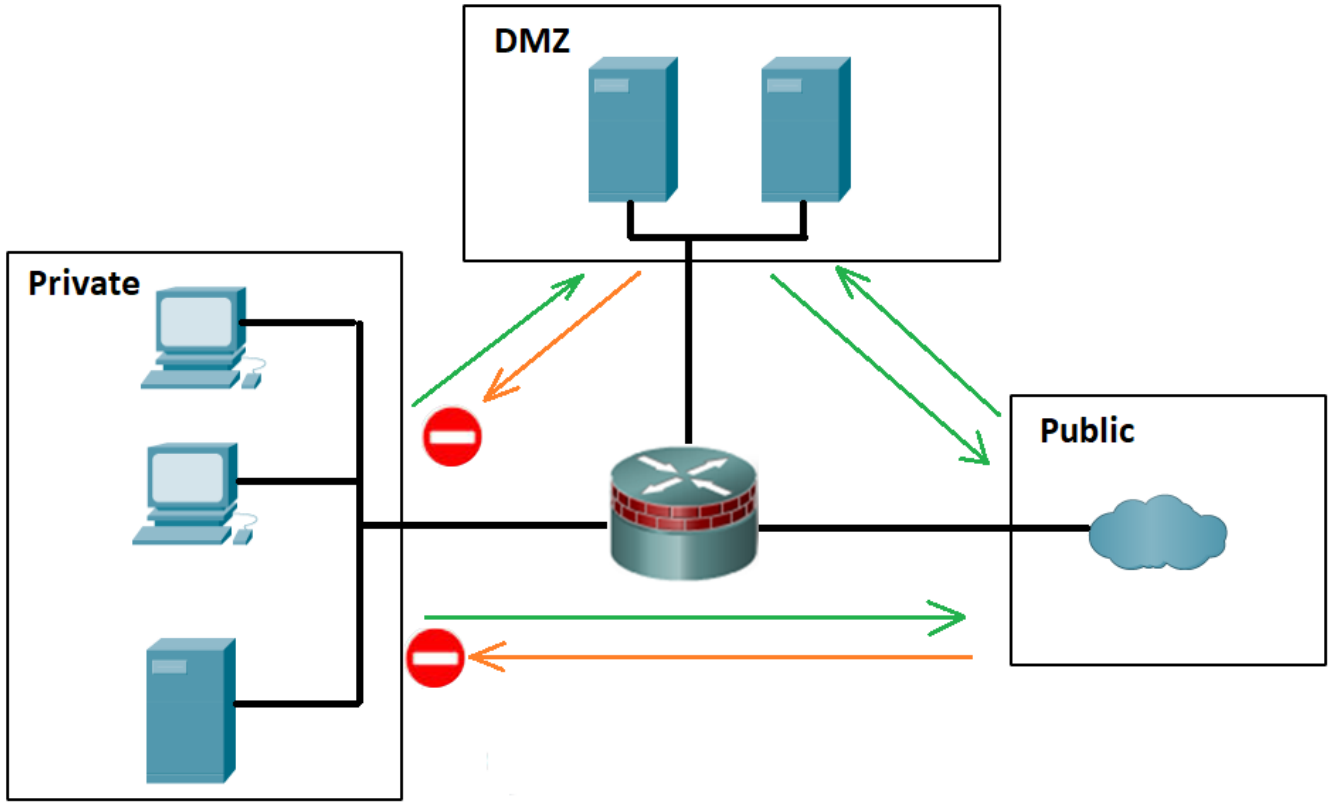
Stateful Firewall

- Stateful firewalls are capable of monitoring and detecting states of all traffic on a network to track and defend based on traffic patterns and flows.
- Stateful firewalls can detect and block attempts by unauthorized individuals to access a network.



Firewall as a core of DMZ

DMZ or demilitarized zone is a physical or logical subnetwork that contains and exposes an organization's external-facing services to an untrusted, public network such as the Internet.



The purpose of a DMZ is to add an additional layer of security to an organization’s private LAN: an external network node can access only what is exposed in the DMZ, while the rest of the organization's network is protected behind a firewall

Firewall Best Practices

- Position firewalls at security boundaries.
- Firewalls are the primary security device, but you shouldn't rely solely on firewalls for security.
- Should deny all traffic except only those services that are needed.
- Regularly monitor firewall logs.
- Primarily protect from technical attacks originating from the outside.

Linux Firewall overview

Linux system firewalls

- **IPtables** - is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules.
- **UFW** (**U**ncomplicated **F**ire**W**all) - is a frontend for managing firewall rules in Arch Linux, Debian, or Ubuntu. UFW is **built upon** IPtables.
- IPtables is a very flexible tool but it's more complex as compared to UFW, it requires a deeper understanding of TCP/IP.

Iptables description

- Iptables - administration tool for IPv4 packet filtering and NAT
- Iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel.
- Several different **tables** may be defined.
- Each table contains a number of built-in **chains** and may also contain user-defined chains.
- Each chain is a list of **rules** which can match a set of packets.
- Each rule specifies what to do with a packet that matches, this is called a '**target**'.

Tables overview

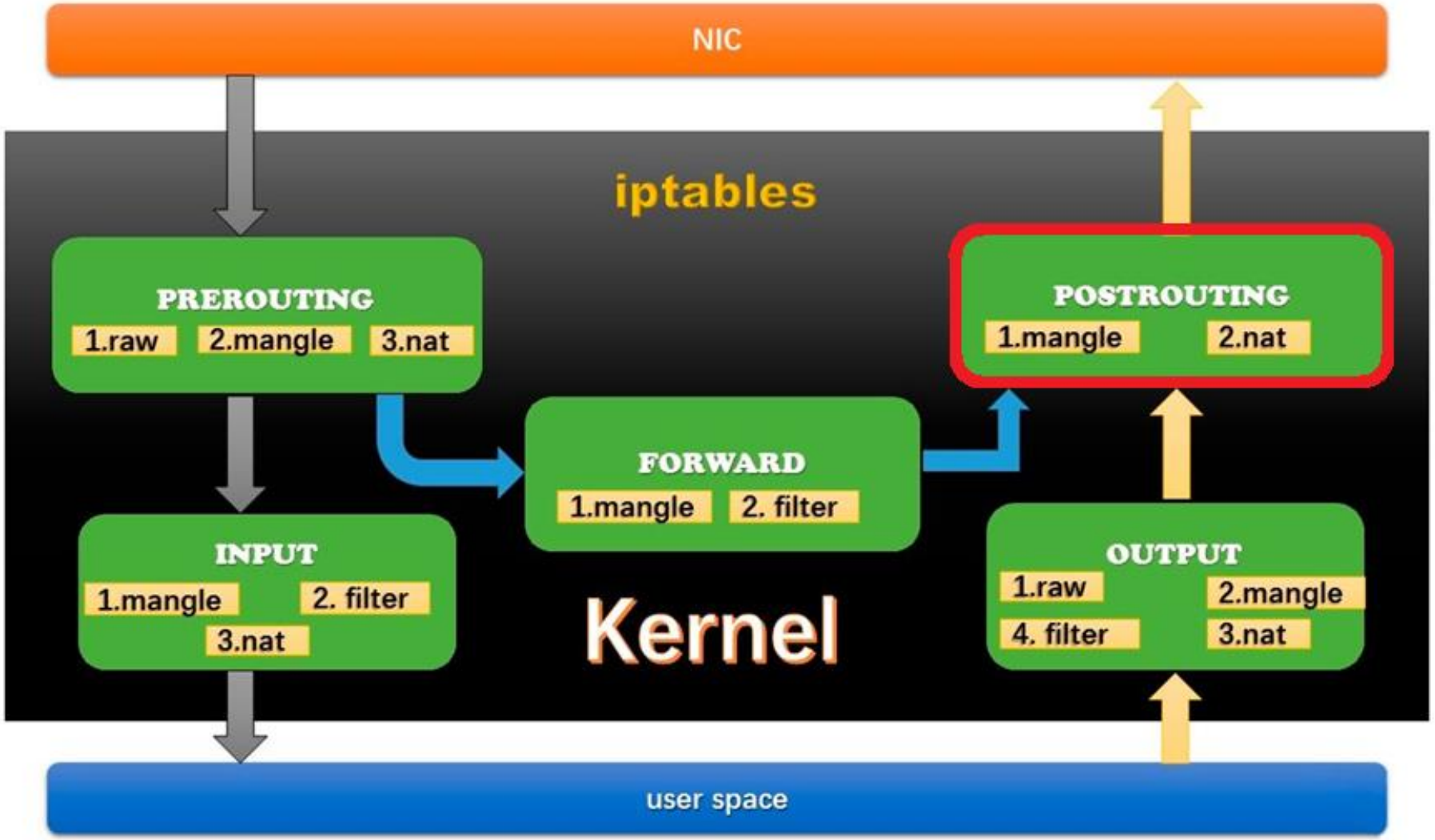
The **filter** table is used to make decisions about whether to let a packet continue to its intended destination or to deny its request.

The **nat** table is used to implement network address translation rules. As packets enter the network stack, rules in this table will determine whether and how to modify the packet's source or destination addresses in order to impact the way that the packet and any response traffic are routed.

The **mangle** table is used to alter the IP headers of the packet in various ways. For instance, you can adjust the TTL (Time to Live) value of a packet, either lengthening or shortening the number of valid network hops the packet can sustain.

The **raw** table has a very narrowly defined function. Its only purpose is to provide a mechanism for marking packets in order to opt-out of connection tracking. The iptables firewall is stateful, meaning that packets are evaluated in regards to their relation to previous packets.

Iptables Process Flow



Targets

- A firewall rule specifies criteria for a packet, and a target.
- If the packet does not match, the next rule in the chain is the examined;
- If the packet does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain or one of the special values:
  - *ACCEPT* means to let the packet through.
  - *DROP* means to drop the packet on the floor.
  - *QUEUE* means to pass the packet to userspace.
  - *RETURN* means stop traversing this chain and resume at the next rule in the previous (calling) chain.

Iptables configuration

Iptables review

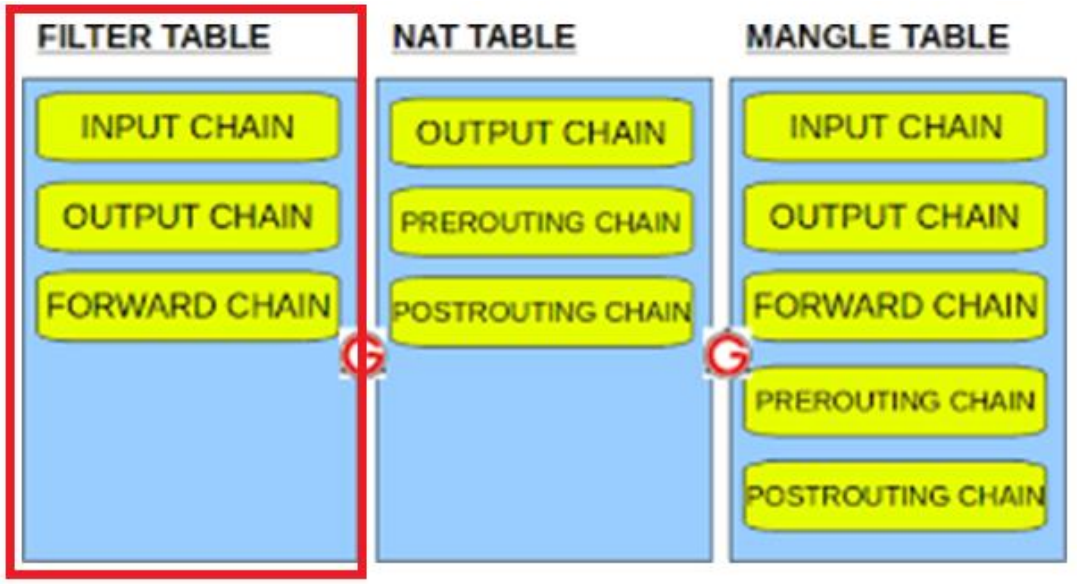
- To check filter tables: *sudo iptables -t <table> -L*

```
sergey@Server1:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
sergey@Server1:~$
```

```
sergey@Server1:~$ sudo iptables -t nat -L
[sudo] password for sergey:
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
sergey@Server1:~$ sudo iptables -t mangle -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

Filter table chains

- The chain names indicate which traffic the rules in each list will be applied to:
  - *input* is for any connections coming to your server;
  - *output* is any leaving traffic;
  - *forward* for any pass through.
- There are two rules are used in filter chains: *accept* and *drop*
- Each chain also has its *policy* setting which determines how the traffic is handled if it doesn't match any specific rules, by default it's set to *accept*.




```
sergey@Server1:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
sergey@Server1:~$
```

Adding rules

- Firewalls can commonly be configured in one of two ways:
    1. set the default rule to accept and then block any unwanted traffic with specific rules;
    2. using the rules to define allowed traffic and blocking everything else;
- The second is often the recommended approach, as it allows pre-emptively blocking traffic, rather than having to reactively reject connections that should not be attempting to access your server.
- To begin using iptables, you should first add the rules for allowed inbound traffic for the services you require.
- Iptables can **track the state of the connection**, so use the command below to allow established connections to continue:

sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

```
sergey@Server1:~$ sudo iptables -L
[sudo] password for sergey:
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,EST
ABLISHED
```

Allowing Incoming Traffic on Specific Ports

To allow incoming traffic on the default SSH port (22):

sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT

```
osboxes@Server1:~$ sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
osboxes@Server1:~$ sudo iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTA
BLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
osboxes@Server1:~$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
osboxes@Server1:~$ sudo iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTA
BLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
osboxes@Server1:~$
```

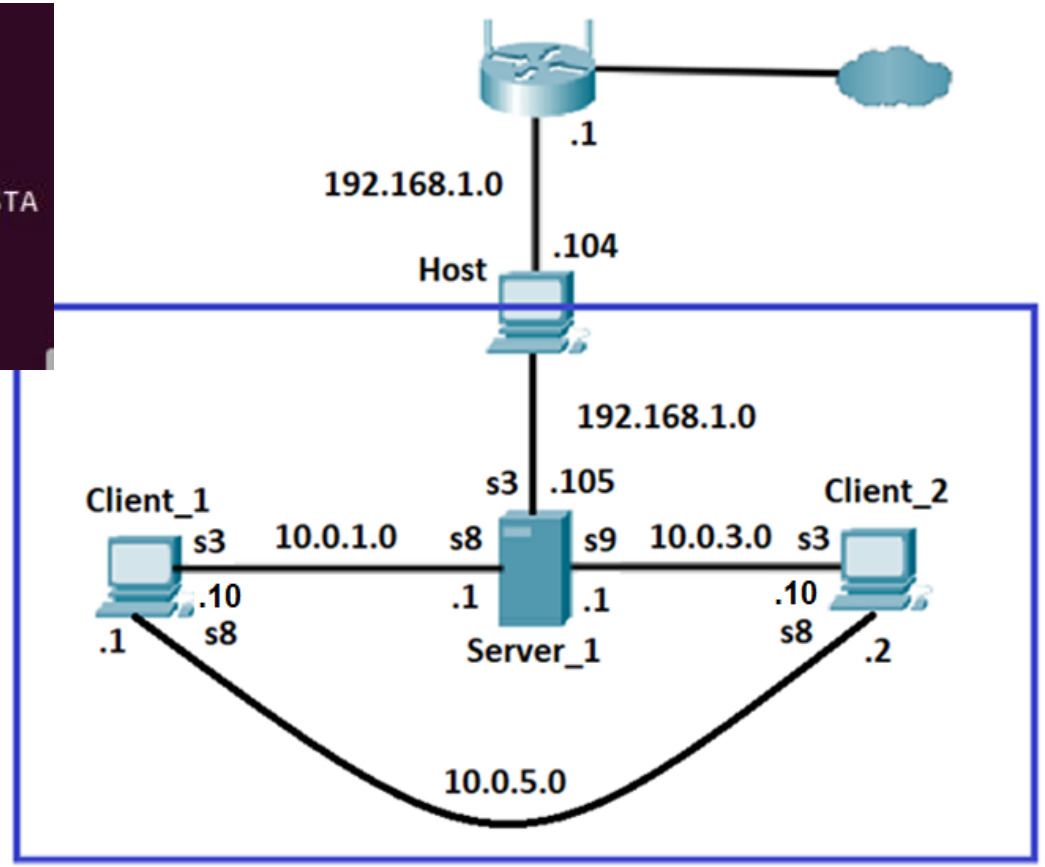
Blocking Traffic

```
osboxes@Client1:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.759 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=1.22 ms
```

sudo iptables -A INPUT -j DROP

```
osboxes@Server1:~$ sudo iptables -A INPUT -j DROP
osboxes@Server1:~$ sudo iptables -L INPUT
sudo: unable to resolve host Server1: Temporary failure in name resolution
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTA
BLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
DROP       all  --  anywhere             anywhere
```

```
osboxes@Client1:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
^C
--- 10.0.1.1 ping statistics ---
14 packets transmitted, 0 received, 100% packet loss, time 13306ms

osboxes@Client1:~$ ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=64 time=0.534 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=64 time=0.685 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=64 time=0.625 ms
```

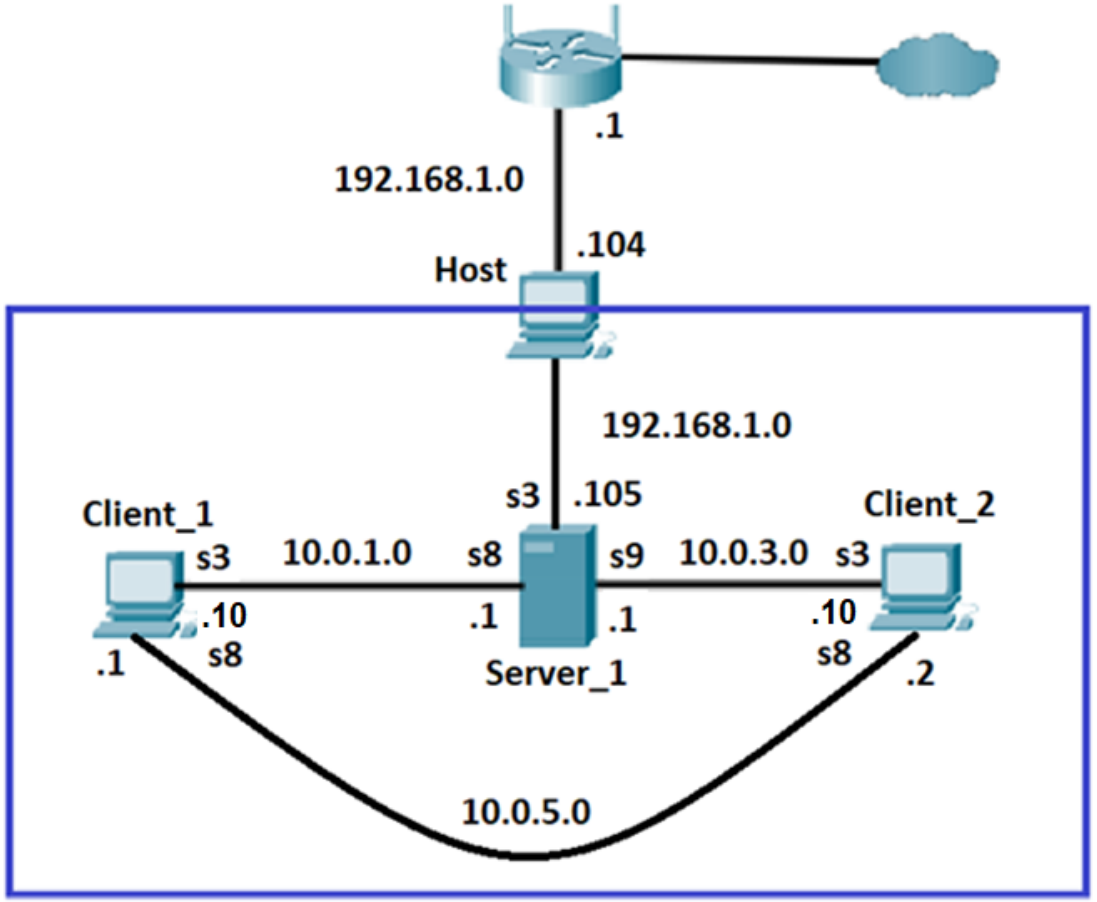


Editing iptables

```
osboxes@Server1:~$ sudo iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere             state RELATED
,ESTABLISHED
2    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
3    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
4    DROP       all  --  anywhere             anywhere
osboxes@Server1:~$ sudo iptables -I INPUT 2 -p icmp -j ACCEPT
osboxes@Server1:~$ sudo iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere             state RELATED
,ESTABLISHED
2    ACCEPT     icmp --  anywhere             anywhere
3    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
4    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
5    DROP       all  --  anywhere             anywhere
osboxes@Server1:~$
```

```
osboxes@Client1:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.525 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=1.61 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=1.34 ms
^C
```

sudo iptables -I INPUT 2 -p icmp -j ACCEPT



iptables -R, --replace chain rulenum rule-specification

Replace a rule in the selected chain. Rules are numbered starting at 1.

Deleting rule from iptables

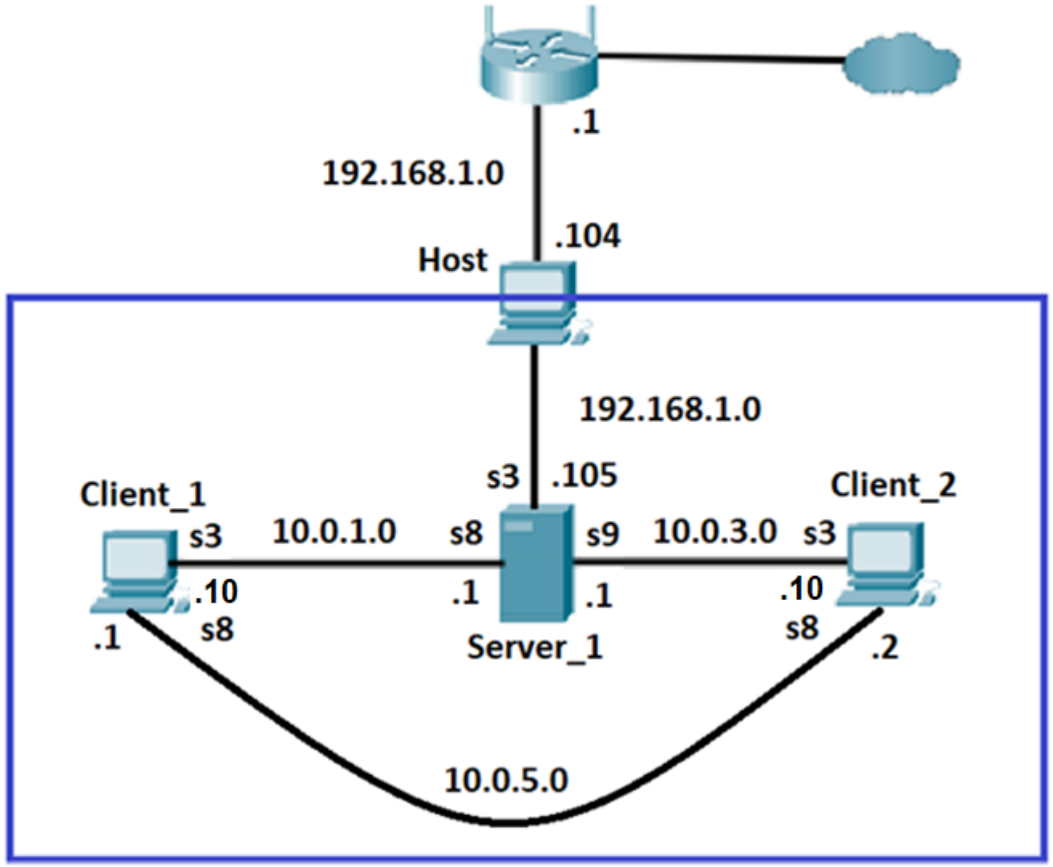
There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

```
osboxes@Server1:~$ sudo iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere             state RELATED
,ESTABLISHED
2    ACCEPT     icmp --  anywhere             anywhere
3    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
4    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
5    DROP       all  --  anywhere             anywhere
osboxes@Server1:~$ sudo iptables -D INPUT 2
osboxes@Server1:~$ sudo iptables -L INPUT --line-numbers
sudo: unable to resolve host Server1: Temporary failure in name resolution
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere             state RELATED
,ESTABLISHED
2    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
3    ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
```

```
osboxes@Client1:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
^C
--- 10.0.1.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3056ms
```

sudo iptables -D INPUT 2

sudo iptables -D INPUT -p icmp -j ACCEPT



Source and destination iptables identification

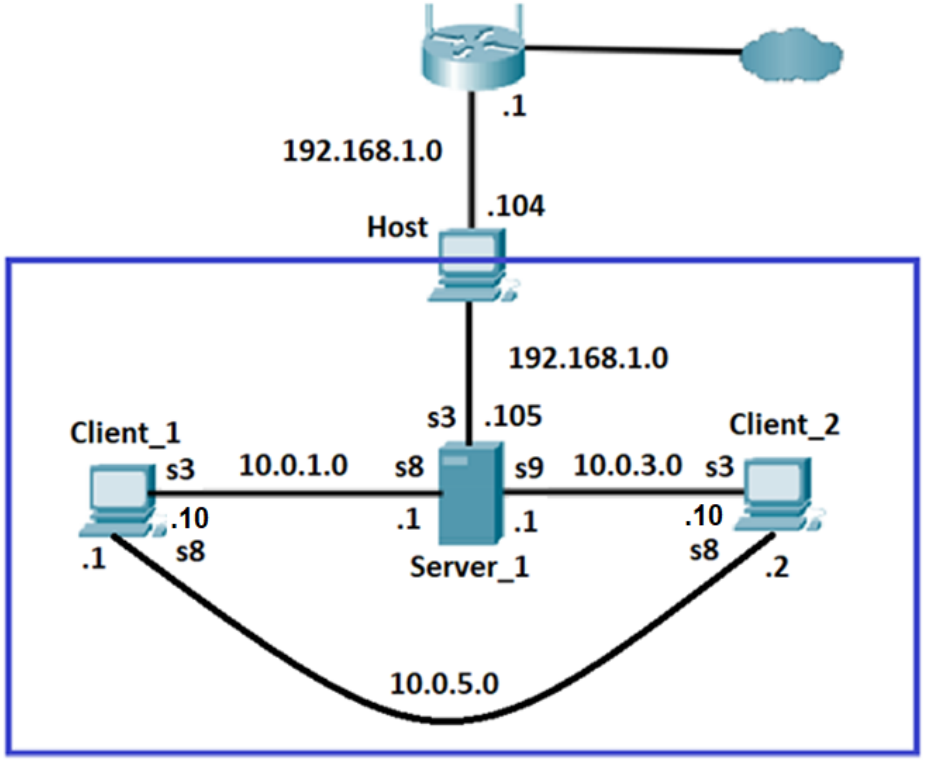
-s, --source [!] address[/mask]

-d, --destination [!] address[/mask]

Source(destination) specification:

- Address can be either a network name, a hostname, a network IP address (with /mask), or a plain IP address.
- The mask can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of 24 is equivalent to 255.255.255.0.

```
osboxes@Server1:~$ sudo iptables -A INPUT -p tcp -s 10.0.1.10/24 --dport 22 -j A
CCEPT
[sudo] password for osboxes:
osboxes@Server1:~$ sudo iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  10.0.1.0/24          anywhere             tcp dpt:ssh
osboxes@Server1:~$
```



Some other useful iptables commands

iptables -L –v

```
osboxes@Server1:~$ sudo iptables -L -v
sudo: unable to resolve host Server1: Temporary failure in name resolution
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

  415 45371 ACCEPT     all  --  any    any     anywhere             anywhere
        state RELATED,ESTABLISHED
    1    84 ACCEPT     icmp --  any    any     anywhere             anywhere

    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
        tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
        tcp dpt:http
  978 89480 DROP       all  --  any    any     anywhere             anywhere
```

iptables -Z, --zero [chain]

Zero the packet and byte counters in all chains.

```
osboxes@Server1:~$ sudo iptables -Z INPUT
osboxes@Server1:~$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     all  --  any    any     anywhere             anywhere
        state RELATED,ESTABLISHED
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere

    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
        tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere
        tcp dpt:http
    8   512 DROP       all  --  any    any     anywhere             anywhere
```

iptables -F, --flush [chain]

Flush the selected chain (all the chains in the table if none is given). This is equivalent to deleting all the rules one by one.

```
osboxes@Server1:~$ sudo iptables -F
osboxes@Server1:~$ sudo iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
osboxes@Server1:~$
```

Saving iptables

- If you were to reboot your machine right now, your iptables configuration would **disappear**.
- Rather than type this each time you reboot, however, you can save the configuration, and have it start up automatically.

  apt install iptables-persistent
- After installation, the actual iptables config, by default will be store in files /etc/iptables/rules.v4
- To save the configuration, you can use

  iptables-save > file-name
- To restore the configuration, you can use

  iptables-restore file-name

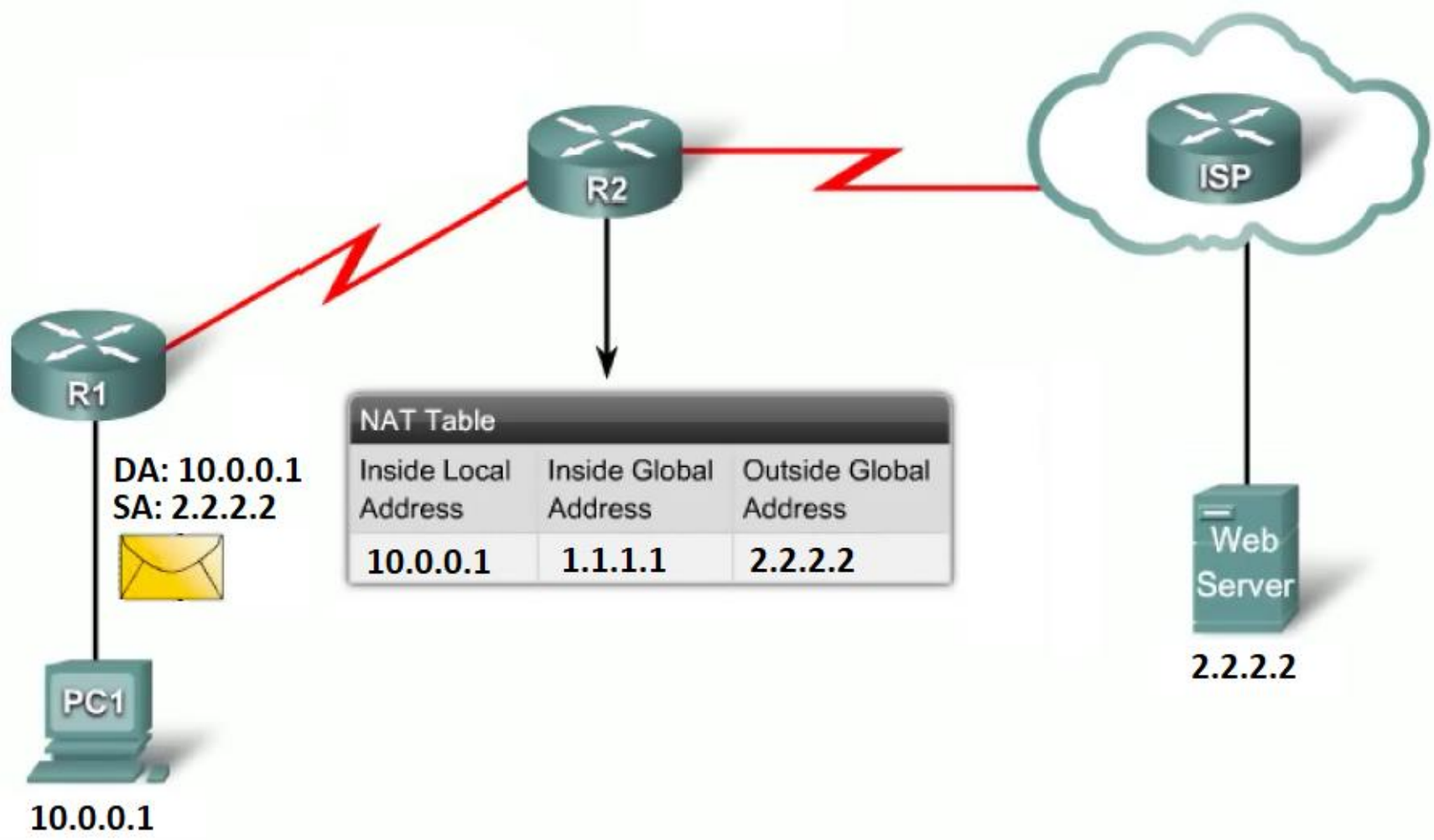
```
sergey@Server1:~$ sudo iptables-restore /etc/iptables/rules.v4
sergey@Server1:~$ iptables -L
Fatal: can't open lock file /run/xtables.lock: Permission denied
sergey@Server1:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
sergey@Server1:~$ cat /etc/iptables/rules.v4
# Generated by iptables-save v1.8.4 on Tue Apr  5 13:10:02 2022
*filter
:INPUT ACCEPT [866:178746]
:FORWARD ACCEPT [843:53972]
:OUTPUT ACCEPT [243:25264]
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
COMMIT
# Completed on Tue Apr  5 13:10:02 2022
```

Linux NAT overview

What is NAT?

- NAT is a process used to **translate** network addresses
- NAT's primary use is to **conserve** public IPv4 addresses
- Usually implemented **at border network devices** such as firewalls or routers
- This allows the networks to use private addresses internally, only translating to public addresses when needed
- Devices within the organization can be assigned private addresses and operate with locally unique addresses.
- Private network address scopes: 10.0.0.0/8, 172.16.0.0/16 to 172.31.0.0/16, 192.168.1.0/24 to 192.168.255.0/24

What is NAT?


R2
ISP
R1
NAT Table
Inside Local Address
Inside Global Address
Outside Global Address
10.0.0.1
1.1.1.1
2.2.2.2
DA: 10.0.0.1
SA: 2.2.2.2
Web Server
2.2.2.2
PC1
10.0.0.1

Types of NAT

- **Static address translation (static NAT) -** One-to-one address mapping between local and global addresses.
- **Dynamic address translation (dynamic NAT) -** Many-to-many address mapping between local and global addresses.
- **Port Address Translation (PAT) -** Many-to-one address mapping between local and global addresses. This method is also known as overloading (NAT overloading).
- **Port Forwarding -** Forwarding a network port from one network node to another
- **Source NAT (SNAT) -** is most commonly used for translating private IP address to a public routable address to communicate with the host. Source NAT changes the source address of the packets that pass through the Router.
- **Destination NAT (DNAT) -** changes the destination address of packets passing through the Router. It also offers the option to perform the port translation in the TCP/UDP headers. Destination NAT mainly used to redirect incoming packets with an external address or port destination to an internal IP address or port inside the network.

PAT Configuration

iptables -t nat -A POSTROUTING -s <net_addr_transl> -j SNAT --to-source <IP_addr_transl>

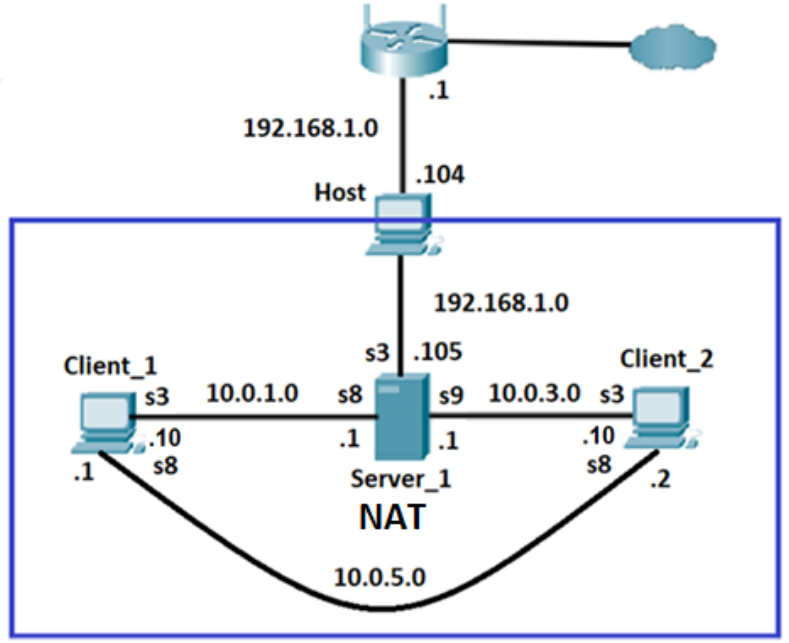
Sample:

iptables -t nat -A POSTROUTING -s 10.0.0.0/16 -j SNAT --to-source 192.168.1.105

```
osboxes@Server1:~$ sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/16 -j SNAT --
to-source 192.168.1.105
osboxes@Server1:~$ sudo iptables -t nat -L POSTROUTING -v
Chain POSTROUTING (policy ACCEPT 1 packets, 64 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 SNAT       all  --  any    any     10.0.0.0/16          anywhere
          to:192.168.1.105
```

```
osboxes@Server1:~$ sudo iptables -t nat -L POSTROUTING -v
Chain POSTROUTING (policy ACCEPT 15 packets, 1154 bytes)
 pkts bytes target     prot opt in     out     source               destination

    8   528 SNAT       all  --  any    any     10.0.0.0/16          anywhere
          to:192.168.1.105
```

iptables -t nat -A POSTROUTING –o <int_name> -j MASQUERADE




```
osboxes@Client1:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=23.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=25.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=25.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=27.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=29.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=25.4 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=32.4 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=20.4 ms
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7012ms
rtt min/avg/max/mdev = 20.412/26.069/32.359/3.433 ms
osboxes@Client1:~$
```

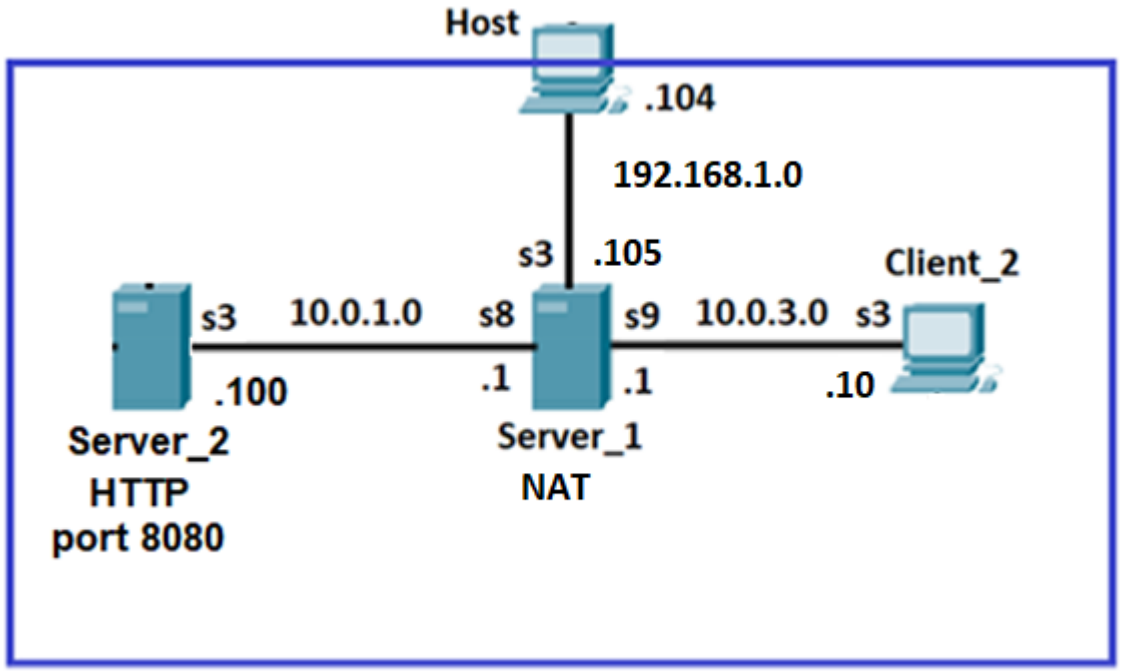
Port Forwarding Configuration

- The first rule specifies that all incoming tcp connections to port 80 should be sent to port 8080 of the internal machine 10.0.1.100.

iptables -A PREROUTING -t nat -i s3 -p tcp --dport 80 -j DNAT --to 10.0.1.100:8080

- The second rule in FORWARD chain allows forwarding the packets to port 8080 of 10.0.1.100.

iptables -A FORWARD -p tcp -d 10.0.1.100 --dport 8080 -j ACCEPT




Thank you!