

Network security basics

Lecture 5.3

Module 5. Linux Networking

Serhii Zakharchenko

Agenda

- Cryptographic services overview
- Remote access configuration
- Firewall technology overview
- Linux Firewall overview
- Q&A

Cryptographic services overview

Cryptographic services for network connectivity

- **Authentication**

- Password method
- Public key method
- Digital signature method

- **Data integrity**

- MD5
- SHA

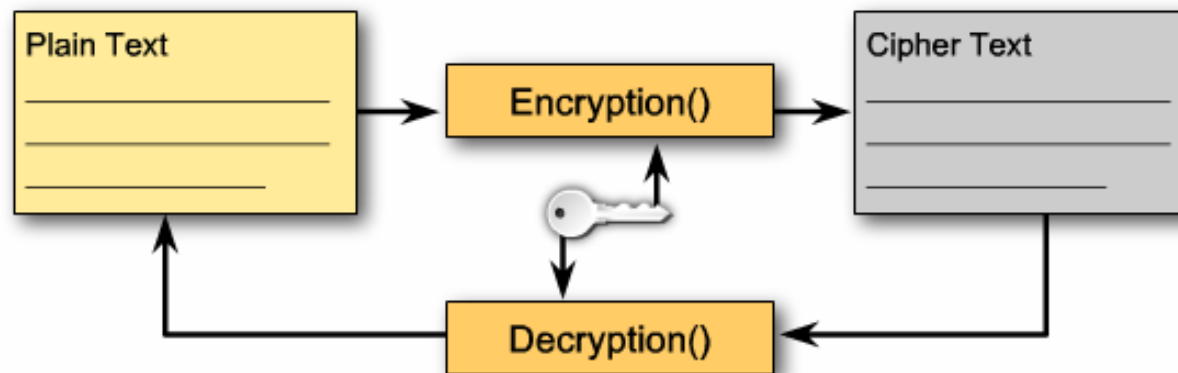
- **Confidentiality**

- DES
- 3DES
- AES

Symmetric and asymmetric algorithms

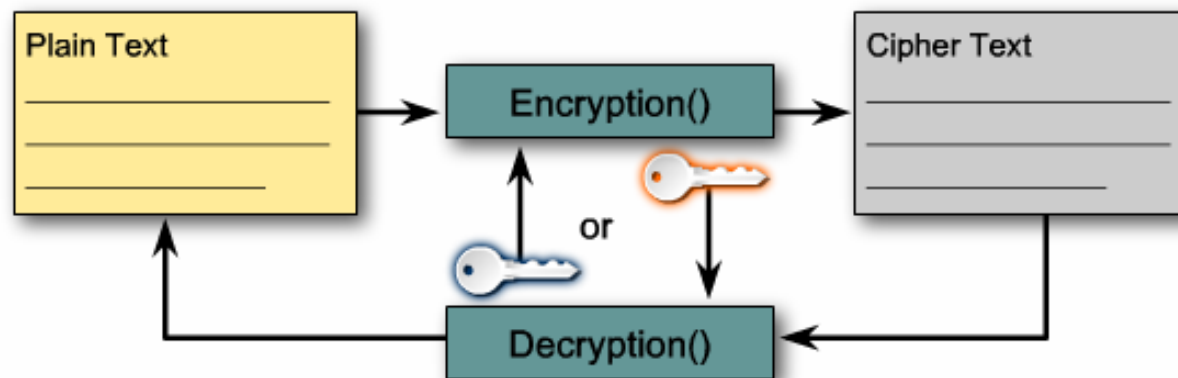
Symmetric algorithm:

- Secret key cryptography
- Encryption and decryption use the same key
- Typically used to encrypt the content of a message
- Examples: DES, 3DES, AES



Asymmetric algorithm:

- Public key cryptography
- Encryption and decryption use different keys
- Typically used in digital certification and key management
- Example: RSA

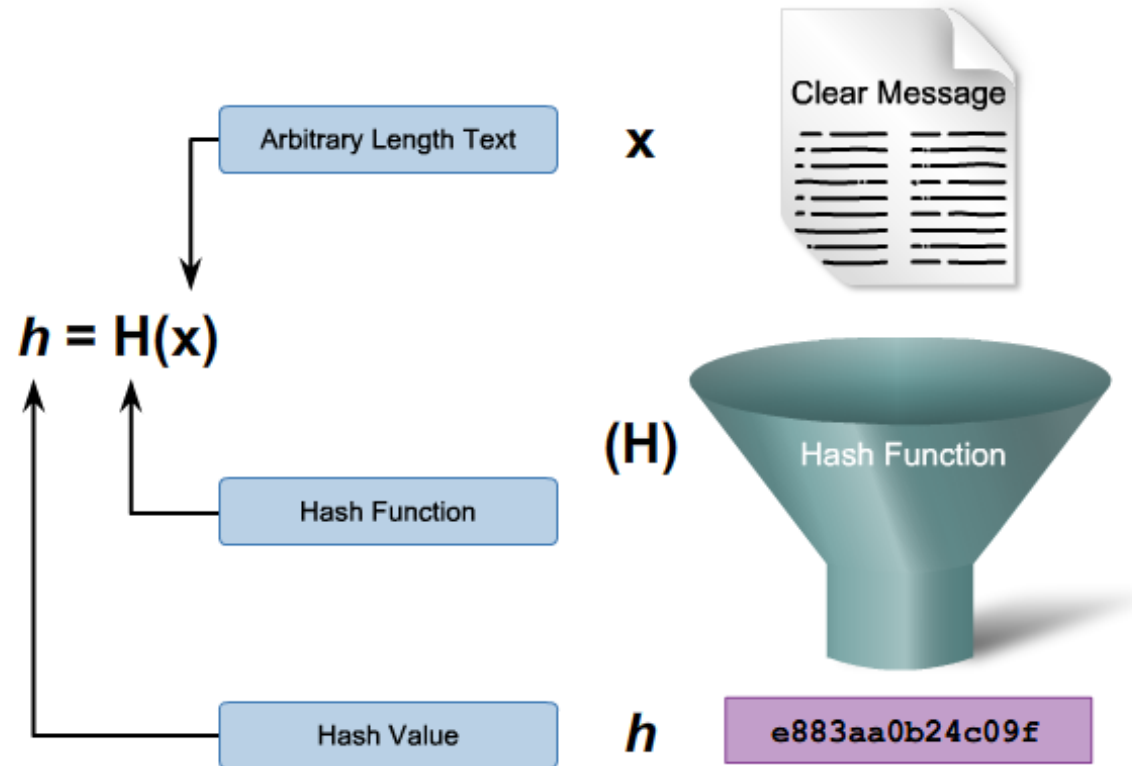


Symmetric and asymmetric algorithms compare

	symmetric	asymmetric
Encryption speed	high	low
Key length	128, 192 i 256	512, 768, 1024, and more
Using	Content encryption	Authentication, Key exchange
Samples	DES, 3DES, AES	RSA

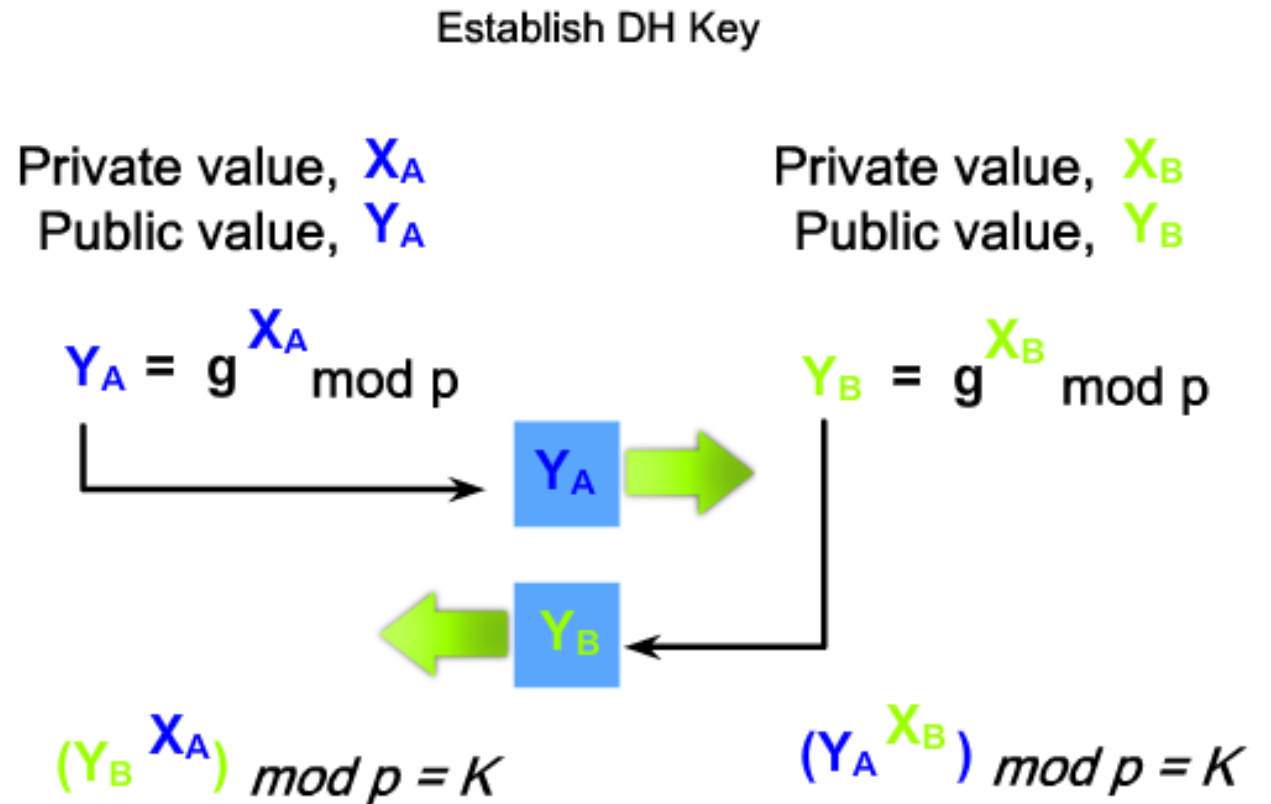
Hash function

- A hash function is any function that can be used to map data of arbitrary size to **fixed-size values**.
- The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.
- Hash functions are one-way functions
- MD (Message Digest): MD2, MD4, MD5. All of them generate digests of a fixed length of **16 bytes**.
- SHA is a modified version of MD5. Examples of SHA names used are SHA-1, SHA-2, SHA-256, SHA-512, SHA-224, and SHA-384,



Diffie-Hellman Algorithm

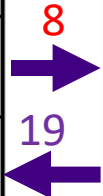
The Diffie–Hellman (DH) Algorithm is a key-exchange protocol that enables two parties communicating over public channel to establish a mutual **secret** without it being transmitted over the Internet.



- In computing, the modulo operation finds the remainder of division of one number by another.
- Given two numbers, X and Y , a modulo p (abbreviated as a mod p) is the remainder, on division of a by p .
- For instance: " $8 \bmod 3$ " would evaluate to 2 ; " $9 \bmod 3$ " would evaluate to 0 .

Diffie-Hellman Algorithm sample

Аліса			Боб		
Shared	Secret	Calc	Shared	Secret	Calc
5, 23			5, 23		
	6	$5^6 \bmod 23 = 8$			
				15	$5^{15} \bmod 23 = 19$
		$19^6 \bmod 23 = 2$			$8^{15} \bmod 23 = 2$

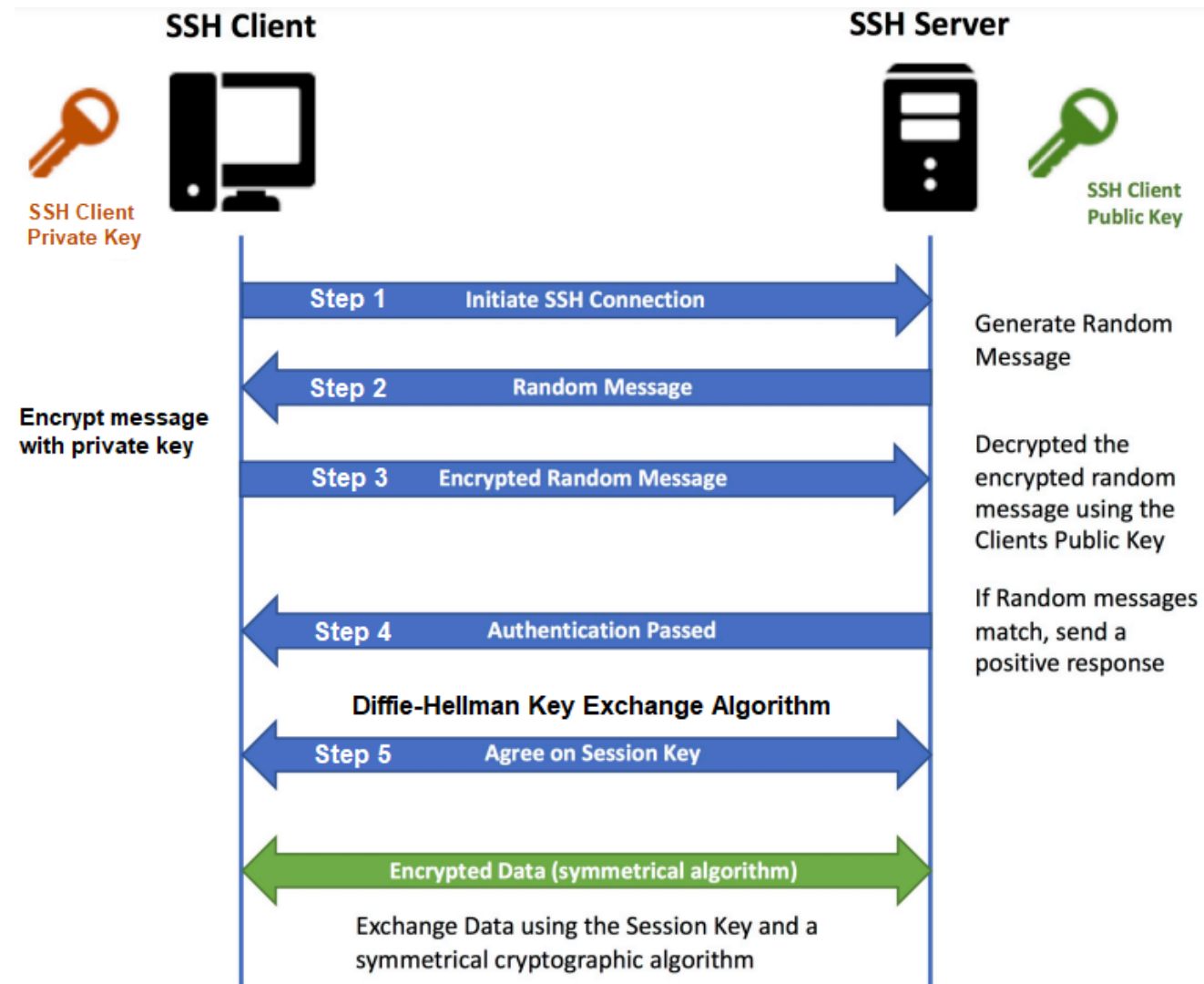


- Bob and Alice agree to use a base number $g=5$ and prime number $p=23$.
 - Alice chooses a secret integer $a=6$.
 - Alice sends Bob $(g^a \bmod p)$ or $5^6 \bmod 23 = 8$.
 - Meanwhile Bob chooses a secret integer $b=15$.
 - Bob sends Alice $(g^b \bmod p)$ or $5^{15} \bmod 23 = 19$.
 - Alice computes $(x^a \bmod p)$ or $19^6 \bmod 23 = 2$.
 - Bob computes $(x^a \bmod p)$ or $8^{15} \bmod 23 = 2$.

Remote access configuration

SSH connection establishing steps

1. SSH Client generates the pair of keys and sends public key to SSH Server.
2. SSH Server generates random message (nonce) and sends it to Client.
3. SSH Client encrypts nonce with private key and sends it to the SSH Server.
4. SSH Server decrypts nonce and compares with sent one. If they matches an authentication is successful.
5. Secret (symmetric) key generates using the Diffie-Hellman Key Exchange Algorithm.



Linux SSH overview

- The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.
- SSH is implemented as two applications - SSH server and SSH client.
- By default, SSH uses TCP 22 port.
- Ubuntu uses a free SSH client and server implementation, OpenSSH.
- When connecting, the client goes through the authorization procedure at the server and an encrypted connection is established between them.
- OpenSSH server can work with both ssh1 and ssh2 protocols, but the ssh1 protocol is considered insecure, so its use is discouraged.

SSH installation

- SSH server installation:

\$ sudo apt install openssh-server

- During installation, the SSH server is automatically assigned to startup. You can control its starting, stopping or restarting using the commands:

\$ sudo systemctl start/stop/restart ssh

- Allow or deny starting the service at system boot:

\$ sudo systemctl enable/disable ssh

- Check that ssh.service is running:

\$ systemctl is-active ssh

```
sergey@Server1:~$ netstat -at -n
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                     LISTEN
tcp6       0      0 :::1:631                 :::*                     LISTEN
sergey@Server1:~$ systemctl is-active ssh
active
sergey@Server1:~$
```

First SSH connecting

- On the first connection, a warning will be issued that the identity of the host to which we are connecting cannot be established:

```
sergey@Client2:/etc/netplan$ ssh zahar@10.0.1.1
The authenticity of host '10.0.1.1 (10.0.1.1)' can't be established.
ECDSA key fingerprint is SHA256:9AEELP7YWEhNSTNgUL+UhaZSe1UfEqhjGfjNjXdl07U.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.1.1' (ECDSA) to the list of known hosts.
zahar@10.0.1.1's password:
```

- After agreeing to connect (yes), the host key will be added to the `~ / .ssh / known_hosts` file.
- And with the next connections, this warning will no longer be.

SSH configuration file

All SSH server settings are stored in the `sshd_config` configuration file located in the `/etc/ssh/` directory. Some configuration recommendations:

- Change the port on which the ssh server is running:

Port 2222

- Deny login as superuser:

PermitRootLogin no

- Disable password authentication (after configuring key authentication):

PasswordAuthentication no

- or if password authentication is still needed, prohibit authorization with an empty password:

PasswordAuthentication yes

PermitEmptyPasswords no

Key authentication config Step1

- The most preferred authentication method is SSH2 RSA key authentication.
- With this method, the user generates on his side a pair of keys, of which one key is secret and the other public.
- The public key is copied to the server and serves to verify the user's identity.

\$ ssh-keygen

- Copying the public key to the server

ssh-copy-id -i RSA_key_client2.pub zahar@10.0.3.1

```
sergey@Client2:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sergey/.ssh/id_rsa): RSA_key_client2
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in RSA_key_client2
Your public key has been saved in RSA_key_client2.pub
The key fingerprint is:
SHA256:hwJ8xcanRJEaLFeCv/RN4CER5AbZzWLPew17ln0n98o sergey@Client2
The key's randomart image is:
+---[RSA 3072]-----+
|      .*=X=o      |
|      .+o0+@ .    |
|      +=@.=       |
|      . ++.= o    |
|      .So + + o   |
|      . o + * =   |
|      . o o+     |
|      . .        |
|      E.         |
+---[SHA256]-----+
sergey@Client2:~$ dir
Desktop  Downloads  Pictures  RSA_key_client2  snap  Videos
Documents Music   Public   RSA_key_client2.pub  Templates
```

```
sergey@Client2:~$ ssh-copy-id -i RSA_key_client2.pub zahar@10.0.3.1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "RSA_key_client2.
pub"
The authenticity of host '10.0.3.1 (10.0.3.1)' can't be established.
ECDSA key fingerprint is SHA256:9AEELP7YWEhNSTNgUl+UhaZSe1UfEqhjGfjNJXdlo7U.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are promp
ted now it is to install the new keys
zahar@10.0.3.1's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'zahar@10.0.3.1'"
and check to make sure that only the key(s) you wanted were added.
```


Key authentication config Step2

- Enable public key authentication and disable password authentication on the server:

*\$ sudo nano
/etc/ssh/sshd_config*

```
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no
```

Key authentication config Step3

- Reboot the ssh service after changing the settings and try to connect to the server using the key:

\$ sudo systemctl restart ssh

- To simple SSH connection add a new entry to ssh_config file

```
sergey@Client2:/etc/ssh$ sudo nano ssh_config
[sudo] password for sergey:
sergey@Client2:/etc/ssh$
```

```
Host server
  User zahar
  HostName 10.0.3.1
  IdentityFile ~/.RSA_key_client2
```

- To connect to the server:

\$ ssh server

```
sergey@Client2:~$ ssh -i RSA_key_client2 zahar@10.0.3.1
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-159-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Oct 11 19:10:57 UTC 2021

System load:  0.0                       Users logged in:      1
Usage of /:   44.0% of 8.79GB           IP address for enp0s3: 192.168.1.105
Memory usage: 16%                       IP address for enp0s8: 10.0.1.1
Swap usage:   0%                         IP address for enp0s9: 10.0.3.1
Processes:   94

70 packages can be updated.
1 update is a security update.

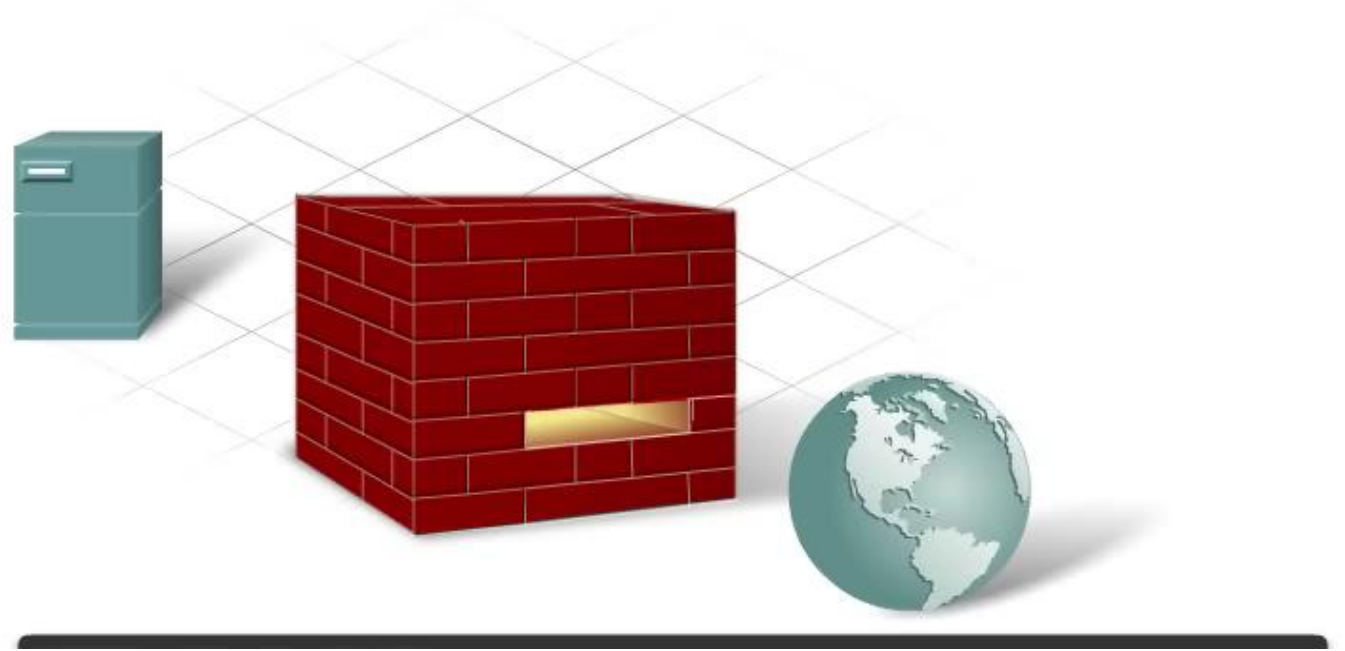
New release '20.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Oct 11 19:08:54 2021 from 10.0.3.10
zahar@Server:~$
```

Firewall technology overview

Firewalls

- A firewall is a system that enforces an access control policy between network
- Common properties of firewalls:
 - The firewall is resistant to attacks
 - The firewall is the only transit point between networks
 - The firewall enforces the access control policy

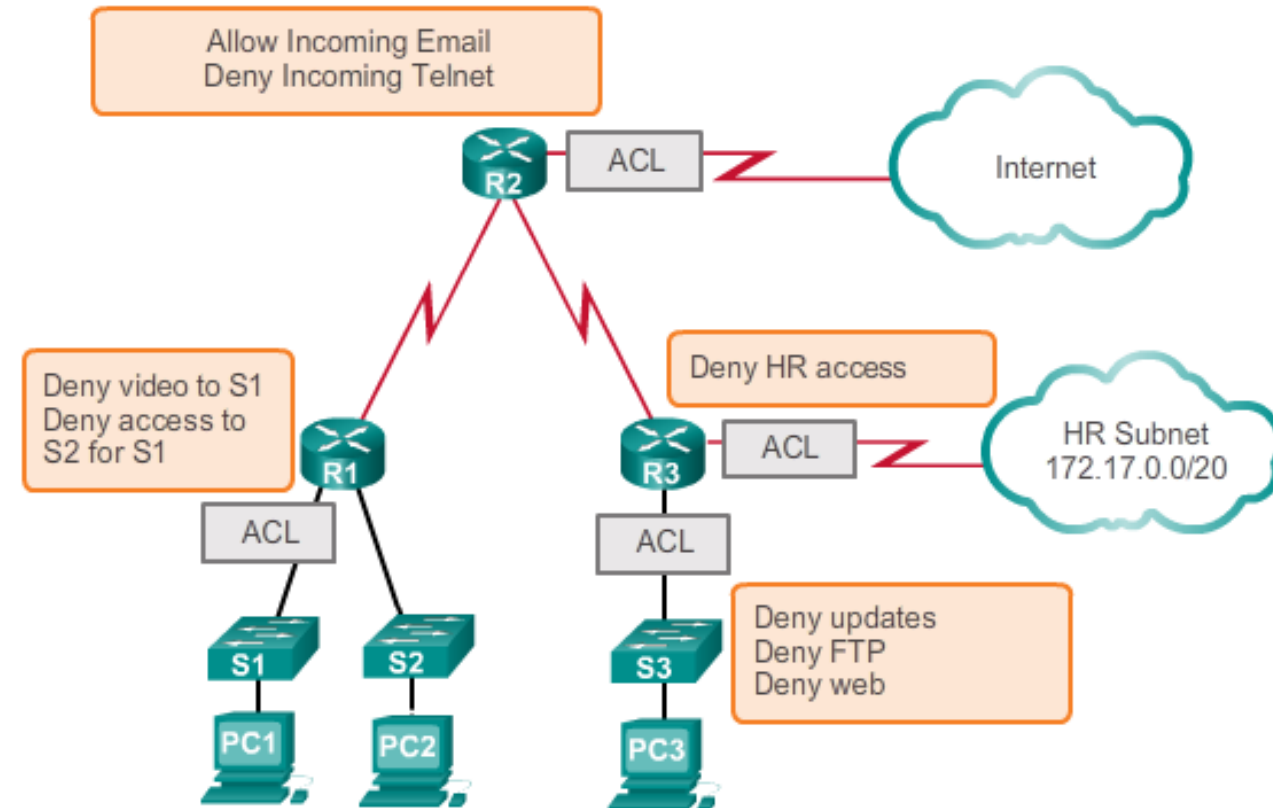


Types of Filtering Firewalls

- **Packet-filtering firewall** —is typically a router that has the capability to filter on some of the contents of packets (examines **Layer 3** and sometimes **Layer 4** information)
- **Stateful (експертний) firewall** —keeps track of the state of a connection: whether the connection is in an initiation, data transfer, or termination state
- **Application gateway firewall (proxy firewall)** —filters information at Layers 3, 4, 5, and 7. Firewall control and filtering done in software.
- **Address-translation firewall** —expands the number of IP addresses available and hides network addressing design.
- **Host-based (server and personal) firewall** —a PC or server with firewall software running on it.
- **Transparent firewall** —filters IP traffic between a pair of bridged interfaces.
- **Hybrid firewalls** —some combination of the above firewalls. For example, an application inspection firewall combines a stateful firewall with an application gateway firewall.

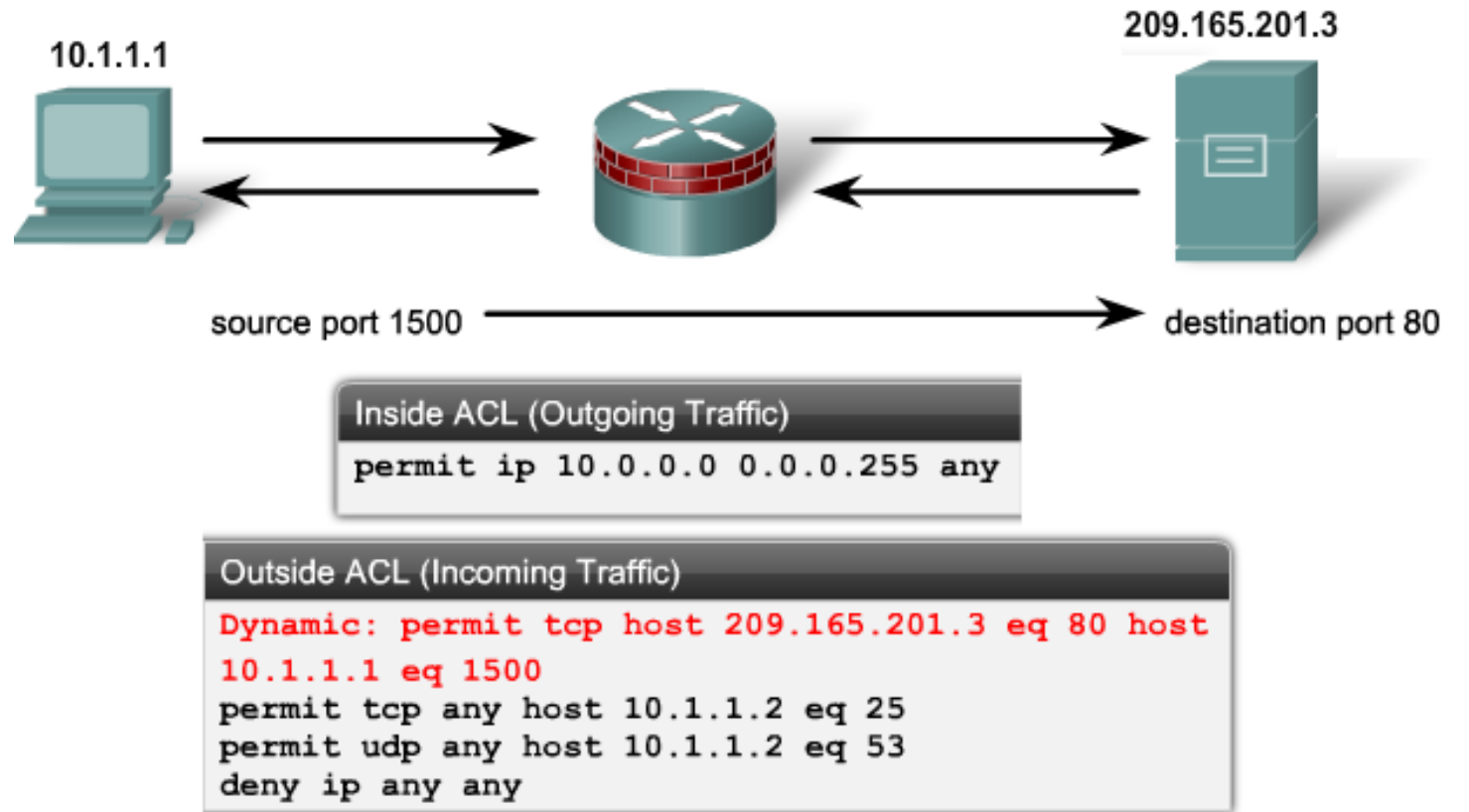
Packet-Filtering Firewall properties

- Are based on simple **permit** or **deny** rule set
- Have a low impact on network performance
- Are easy to implement
- Afford an initial degree of security at a low network layer
- Is susceptible to IP spoofing. Hackers send arbitrary packets that fit ACL criteria and pass through the filter.
- Packet filters are stateless.



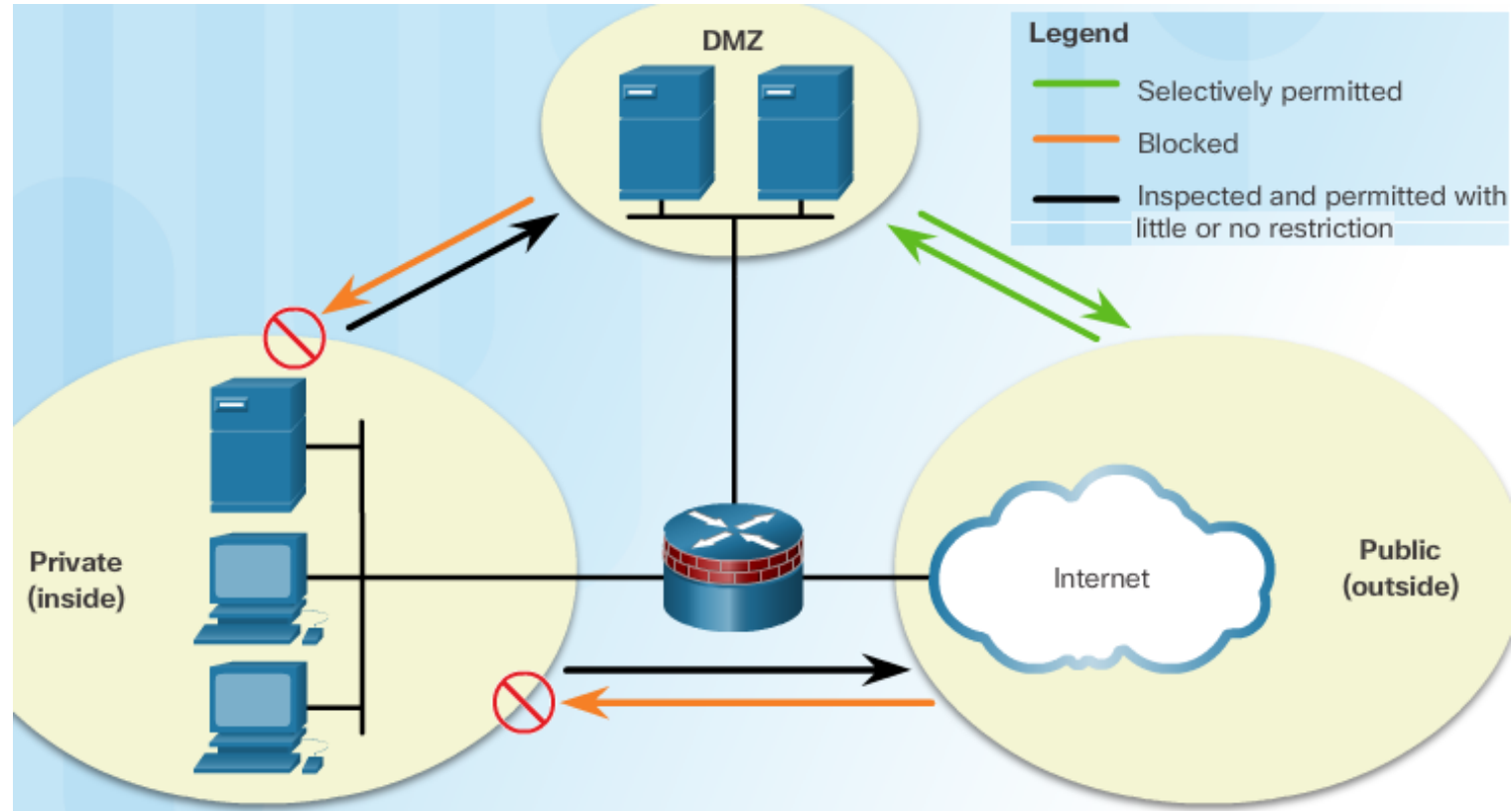
Stateful Firewall vs Stateless

- **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.
- **Stateless** firewalls, however, only focus on individual packets, using preset rules to filter traffic.



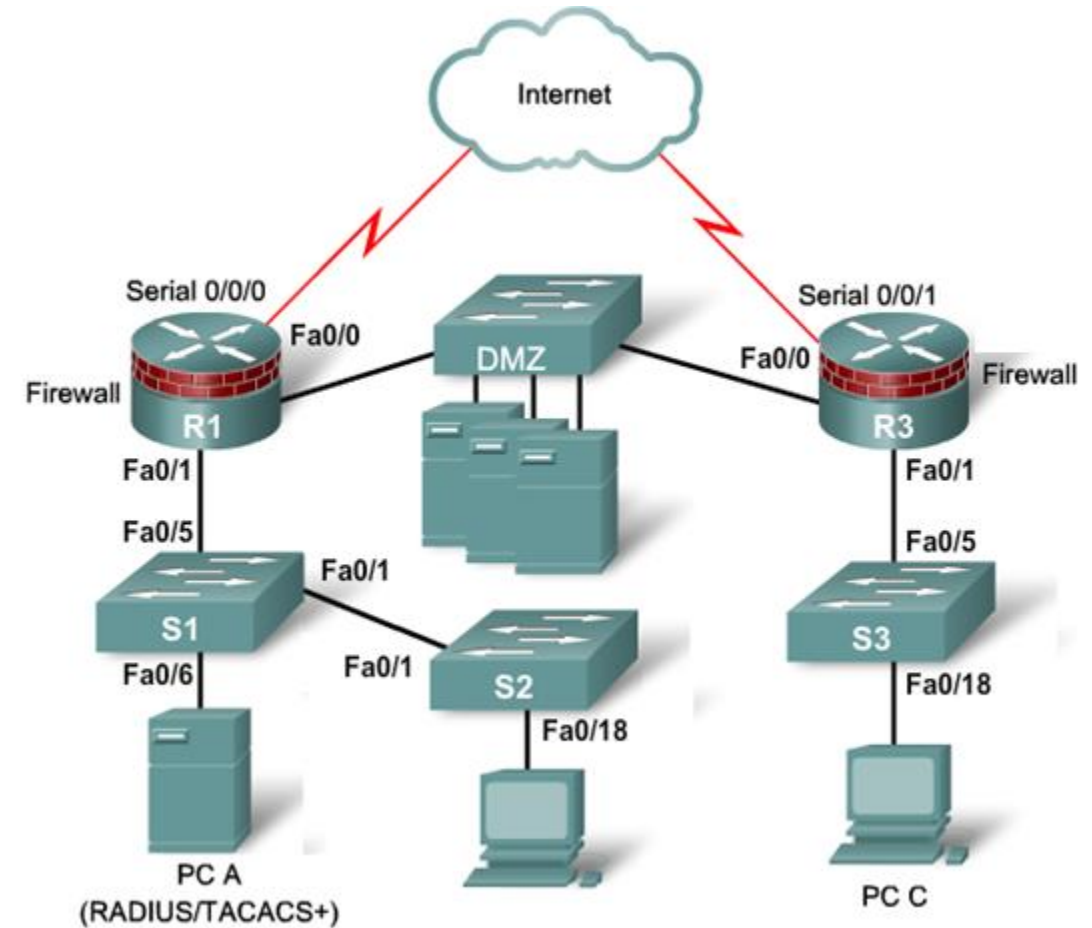
Firewall as a core of DMZ

A DMZ, short for demilitarized zone, is a network (physical or logical) used to connect hosts that provide an interface to an untrusted external network – usually the Internet – while keeping the internal, private network – usually the corporate network – separated and isolated from the external network.



Firewall Best Practices

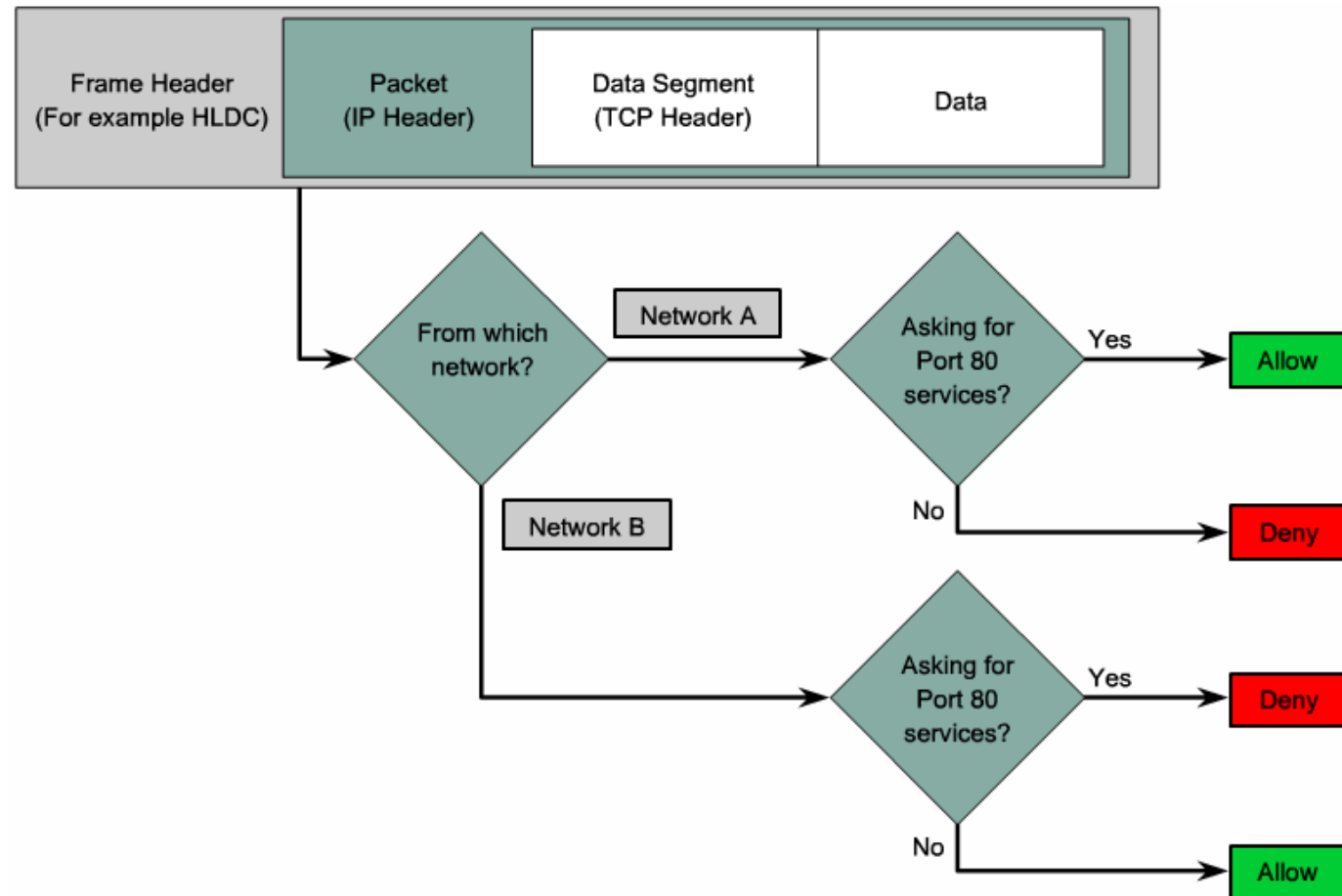
- Position firewalls at security boundaries.
- Firewalls are the primary security device. It is unwise to rely exclusively on a firewall for security.
- Deny all traffic by default. Permit only services that are needed.
- Ensure that physical access to the firewall is controlled.
- Regularly monitor firewall logs.
- Remember that firewalls primarily protect from technical attacks originating from the outside.



Linux Firewall overview

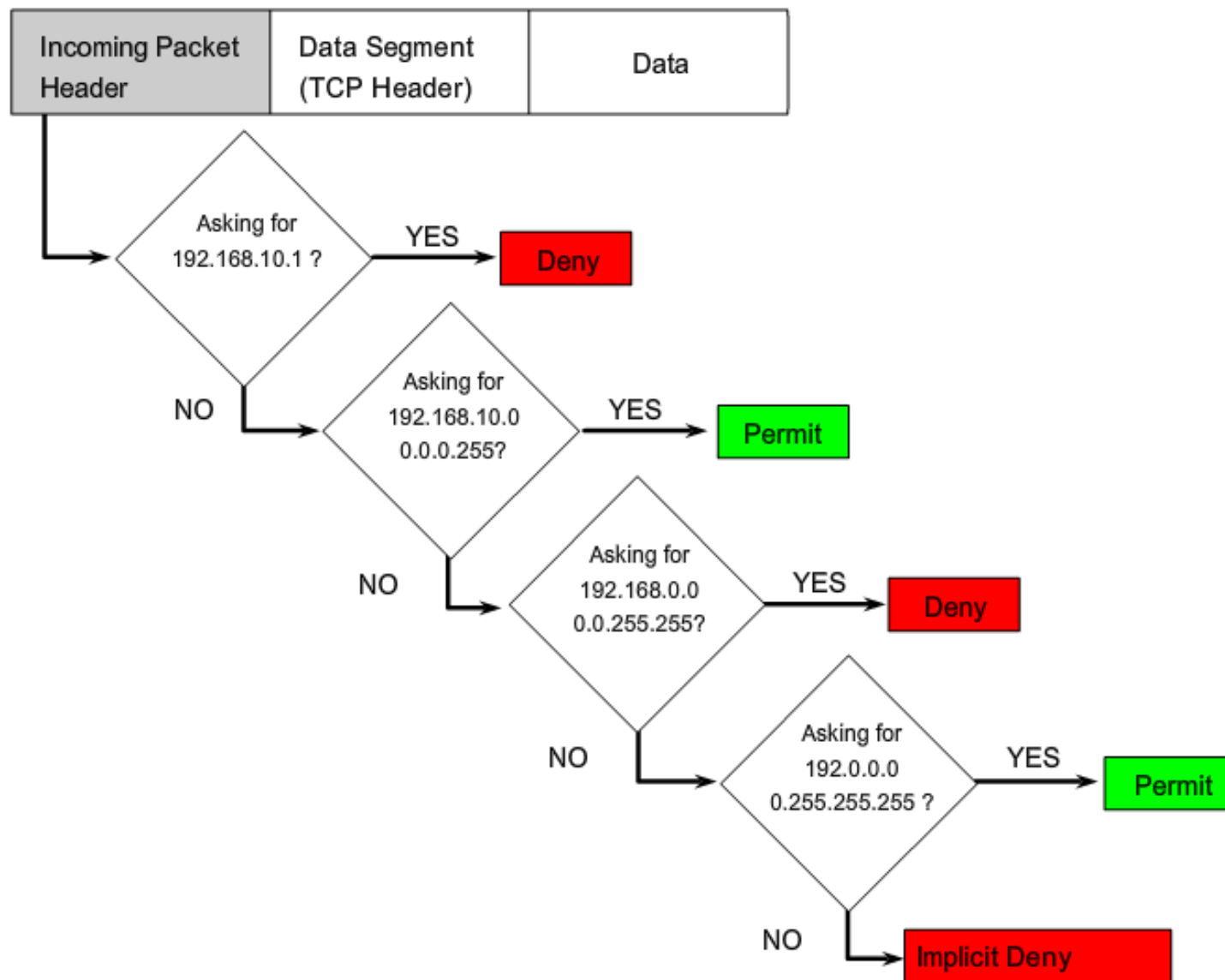
Packet filtering basics

- **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 device (OSI layer 3,4) uses rules to determine whether to permit or deny traffic.
- Packet-filtering device can extract the following information from the packet header:
 - **Source IP address**
 - **Destination IP address**
 - **ICMP message type**
 - **TCP/UDP source port**
 - **TCP/UDP destination port**



Firewall Logic

Rules sequence - from special to general



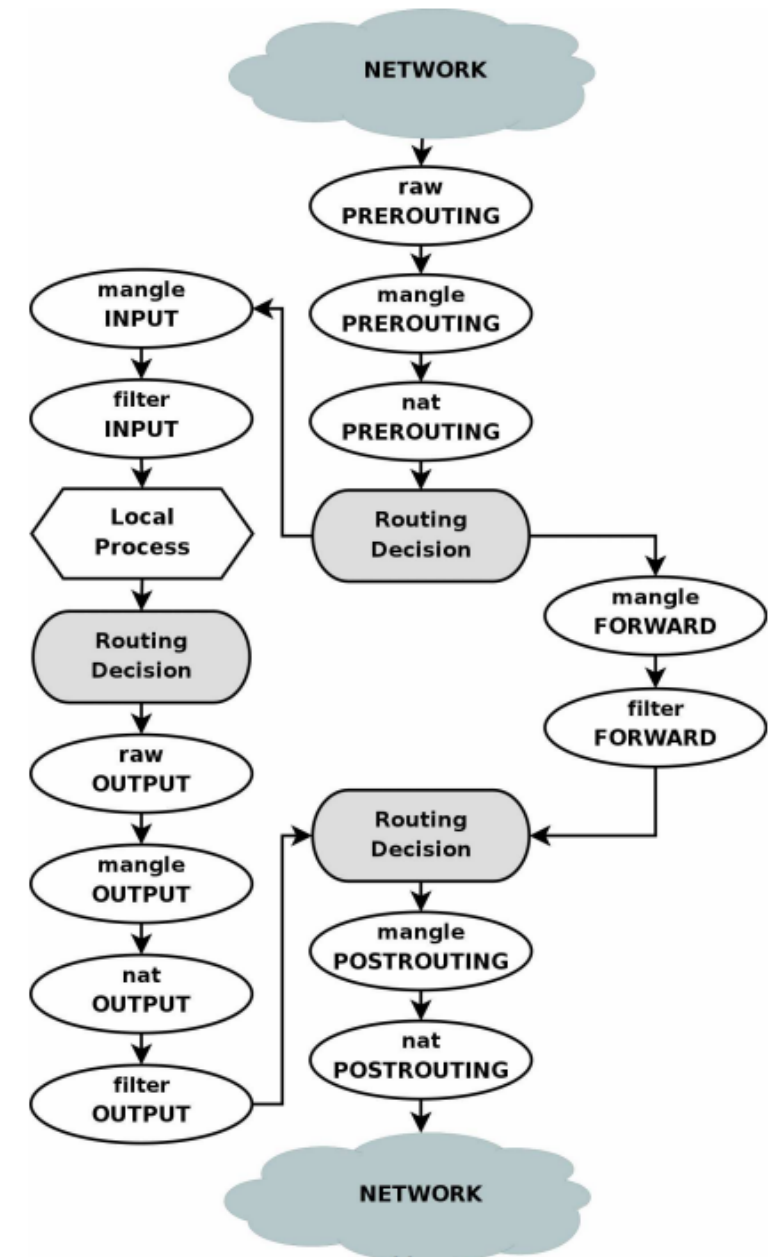
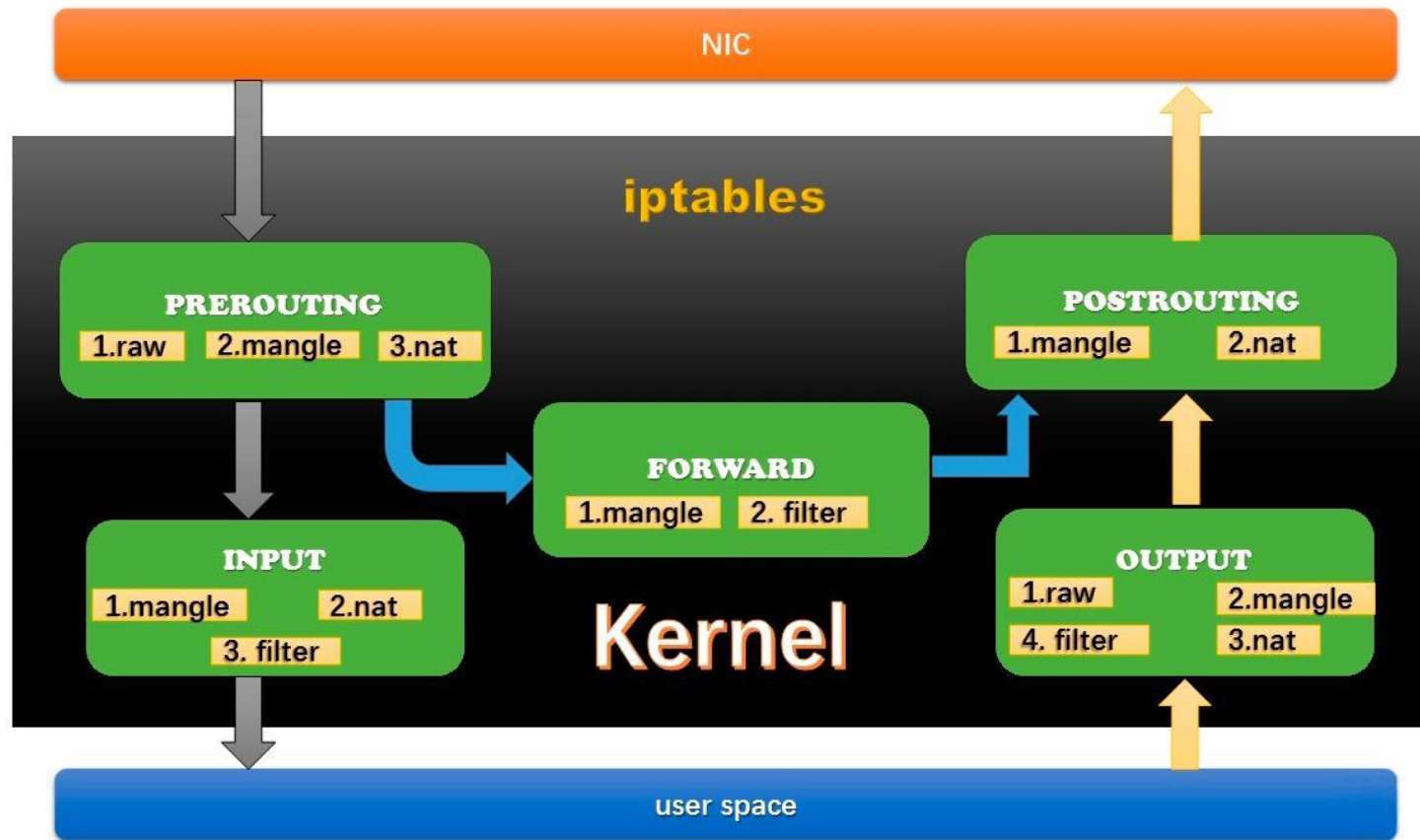
Linux system firewalls

- **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**, or uncomplicated firewall, 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

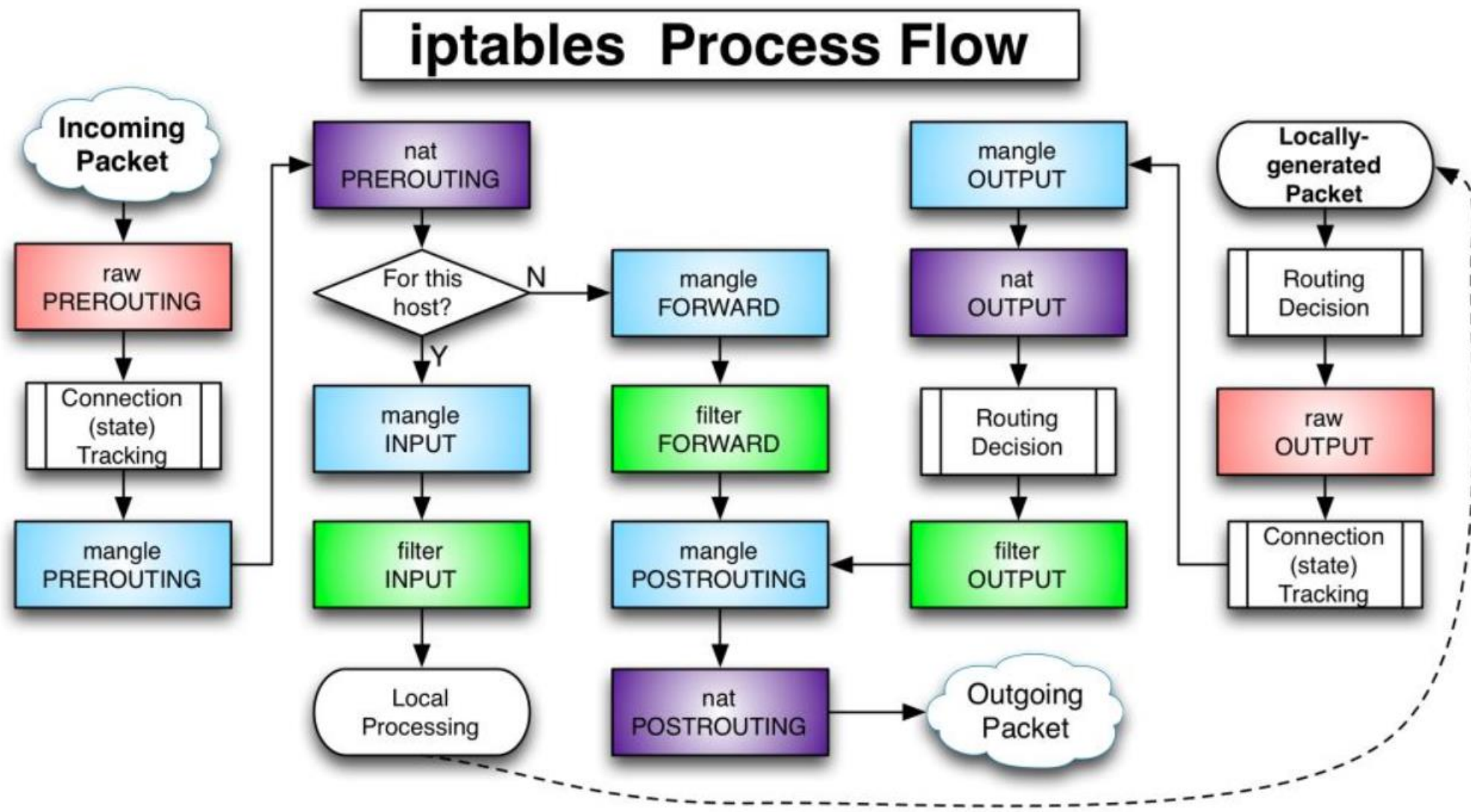
Iptables description

- 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 and chains



Iptables Process Flow



Tables

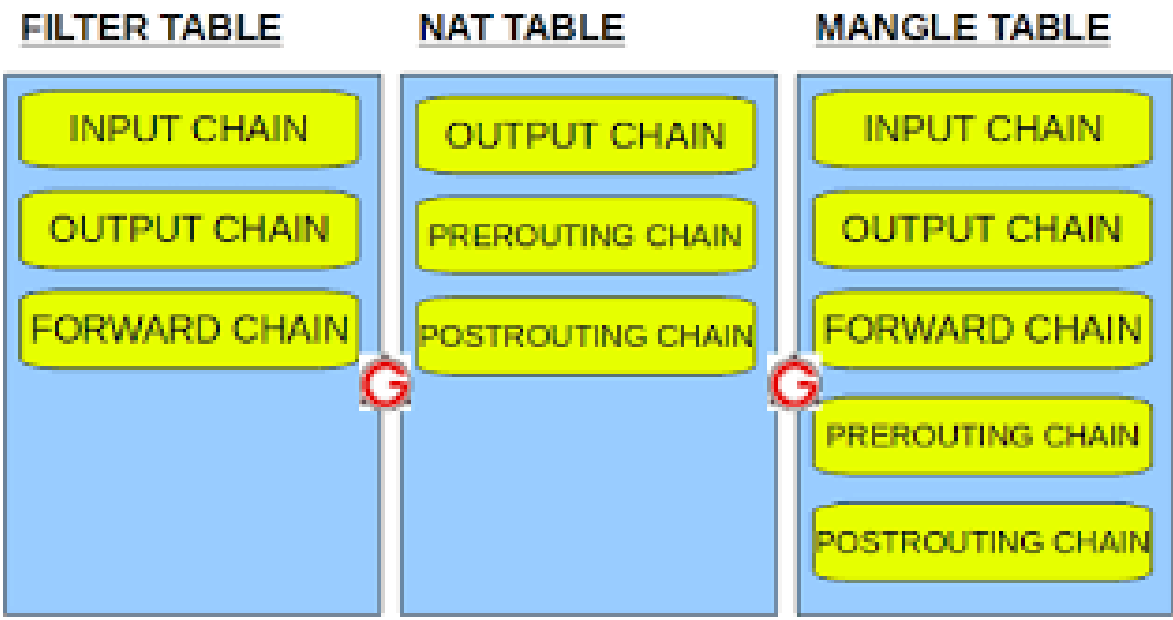
There are currently three main independent tables:

filter: This is the default table (if no -t option is passed).

nat: This table is consulted when a packet that creates a new connection is encountered.

mangle: This table is used for specialized packet alteration.

iptables [-t table] - t option specifies the packet matching table which the command should operate on.



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 review

- Ubuntu servers do not implement any restrictions by default
- 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

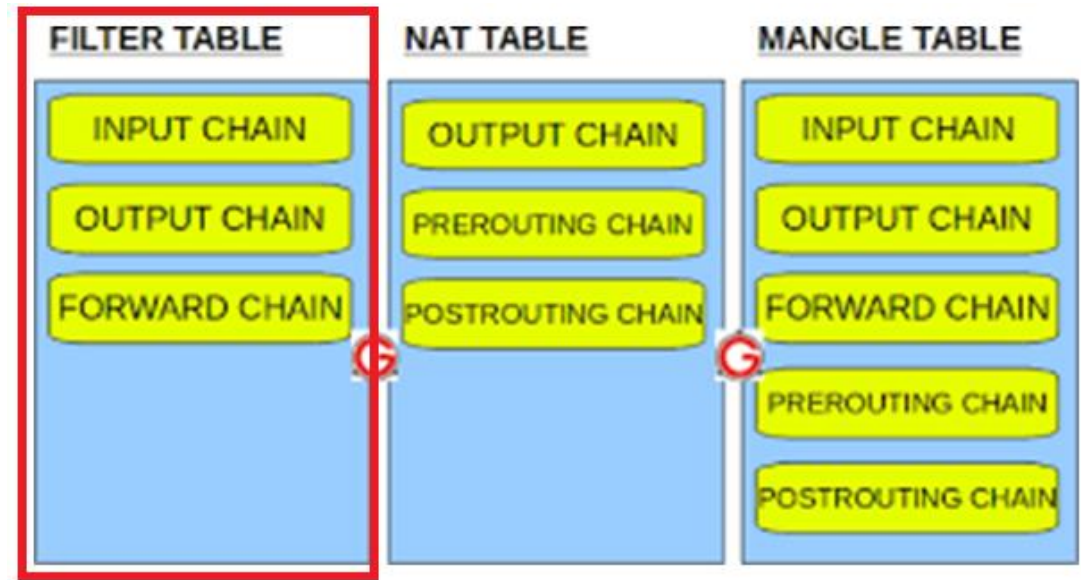
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

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           state RELATED,ESTABLISHED
```

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
```

- append this rule to the input chain (-A INPUT) so we look at incoming traffic
- check to see if it is TCP (-p tcp).
- check to see if the input goes to the SSH port (--dport ssh).
- accept the input (-j ACCEPT).

```
sergey@Server1:~$ sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
[sudo] password for sergey:
sergey@Server1:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            state
ACCEPT     all  --  anywhere              anywhere               state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere              anywhere               tcp dpt:ssh
```

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

Blocking Traffic

Once a decision is made to accept a packet, no more rules affect it. As our rules allowing ssh and web traffic come first, as long as our rule to block all traffic comes after them, we can still accept the traffic we want. All we need to do is put the rule to block all traffic at the end.

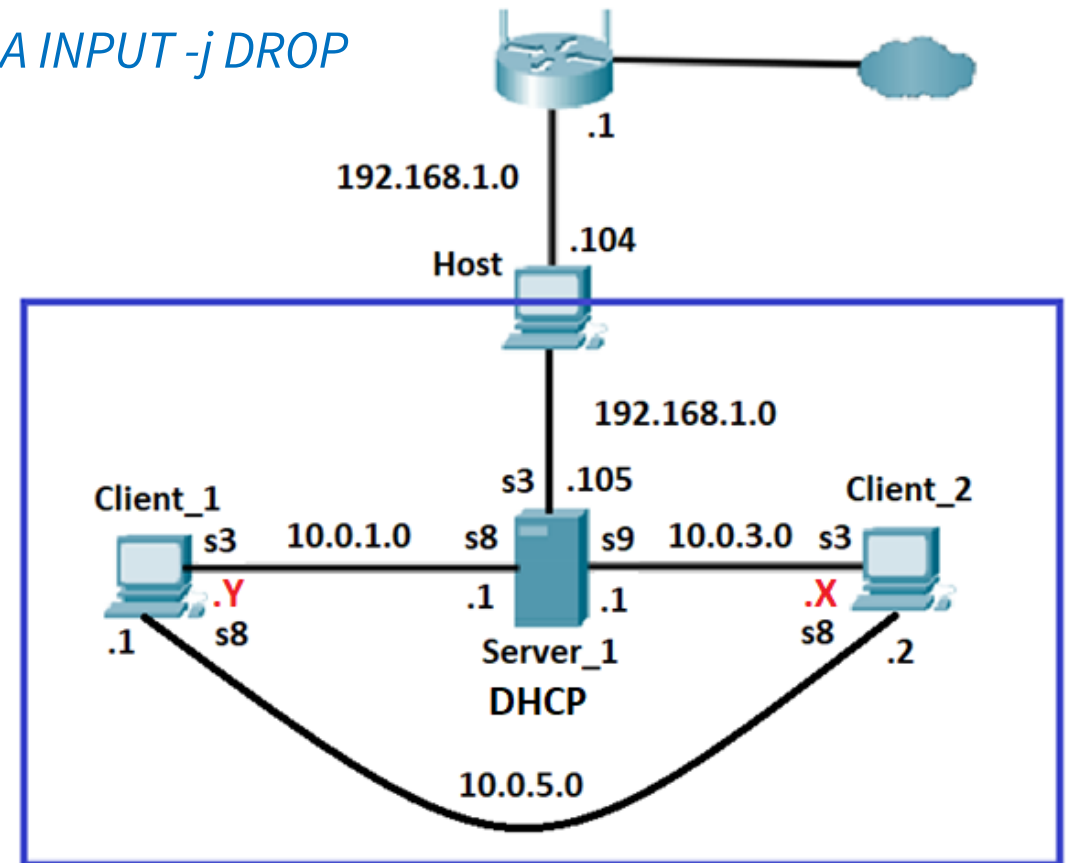
```
sergey@Client1:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=35 ttl=64 time=1.55 ms
64 bytes from 10.0.1.1: icmp_seq=36 ttl=64 time=1.17 ms
64 bytes from 10.0.1.1: icmp_seq=37 ttl=64 time=1.45 ms
```

sudo iptables -A INPUT -j DROP

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

```
sergey@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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4096ms
```

```
sergey@Client1:~$ ping 10.0.3.12
PING 10.0.3.12 (10.0.3.12) 56(84) bytes of data.
64 bytes from 10.0.3.12: icmp_seq=1 ttl=63 time=0.881 ms
64 bytes from 10.0.3.12: icmp_seq=2 ttl=63 time=2.25 ms
64 bytes from 10.0.3.12: icmp_seq=3 ttl=63 time=1.25 ms
^C
```



Editing iptables

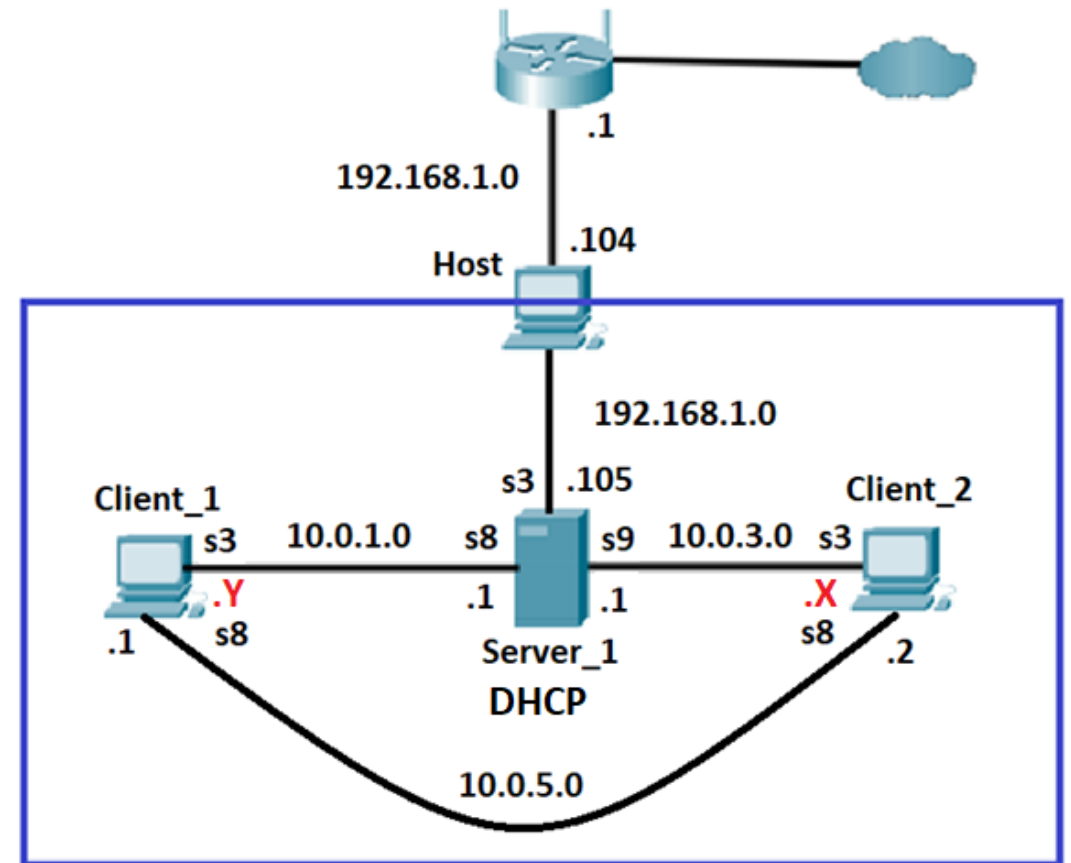
-I, --insert chain [rulenum] rule-specification

Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain. This is also the default if no rule number is specified.

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

```
sergey@Server1:~$ sudo iptables -I INPUT 2 -p icmp -j ACCEPT
sudo: unable to resolve host Server1: Temporary failure in name resolution
[sudo] password for sergey:
sergey@Server1:~$ sudo iptables -L
sudo: unable to resolve host Server1: Temporary failure in name resolution
Chain INPUT (policy ACCEPT)
target     prot opt source               destination          state
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere             anywhere
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
DROP       all  --  anywhere             anywhere
```

```
sergey@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=0.629 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=1.20 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.646 ms
^C
```



Deleting rule from iptables

-D, --delete chain rule-specification

-D, --delete chain rulenum

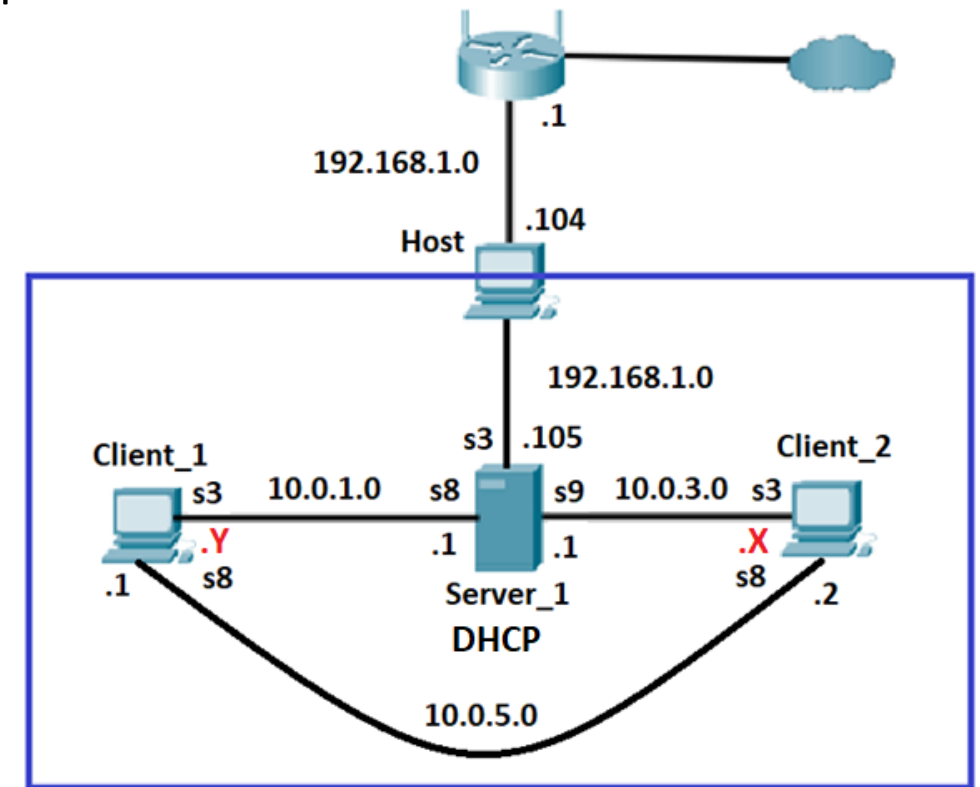
Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

sudo iptables -D INPUT 2

sudo iptables -D INPUT -p icmp -j ACCEPT

```
sergey@Server1:~$ sudo iptables -D INPUT 2
sudo: unable to resolve host Server1: Temporary failure in name resolution
sergey@Server1:~$ sudo iptables -L
sudo: unable to resolve host Server1: Temporary failure in name resolution
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            state
ACCEPT     all  --  anywhere              anywhere               state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere              anywhere               tcp dpt:ssh
DROP       all  --  anywhere              anywhere
```

```
sergey@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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4100ms
```



Some other useful iptables commands

-R, --replace chain rulenum rule-specification

Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.

iptables -L -v

-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.

```
sergey@Server1:~$ sudo iptables -L -v
sudo: unable to resolve host Server1: Temporary failure in name resolution
[sudo] password for sergey:
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source            destination
 11815  62M ACCEPT     all  --  any    any    anywhere          anywhere
      state RELATED,ESTABLISHED
    1    60 ACCEPT     tcp  --  any    any    anywhere          anywhere
      tcp dpt:ssh
 2594  511K DROP       all  --  any    any    anywhere          anywhere
```

-Z, --zero [chain]

Zero the packet and byte counters in all chains. It is legal to specify the -L, --list (list) option as well, to see the counters immediately before they are cleared.

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 (specifying any name to be resolved with a remote query such as DNS is a really bad idea), 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.

```
sergey@Server1:~$ sudo iptables -A INPUT -p tcp -s 10.0.1.12/255.255.255.0 --dport 80 -j ACCEPT
sergey@Server1:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:ssh
ACCEPT     tcp  --  anywhere              anywhere              tcp dpt:ssh
ACCEPT     tcp  --  10.0.1.0/24           anywhere              tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

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

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 and /etc/iptables/rules.v6
- To save the configuration, you can use

iptables-save > file-name

- To restore the configuration, you can use

iptables-restore file-name

```
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
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          tcp dpt:ssh
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
```

UFW peculiarities

- UFW is an acronym for an uncomplicated firewall, as a rule it is installed by default
- UFW is built upon Iptables
- Rules are analyzed line by line – rule's sequence is important.
- UFW doesn't filter ICMP traffic – only traffic based on tcp and udp protocols
- Iptables rules takes a precedence over UFW

UFW initial config

- To view status of ufw:

sudo ufw status

- The default policy firewall closes all ports on the server and open only required ports one by one.
- This allows to block all incoming connections and only allow outgoing connections.

sudo ufw default allow outgoing

sudo ufw default deny incoming

- Allow access to incoming SSH ports:

sudo ufw allow ssh

- Turn on firewall:

sudo ufw enable

- To stop the firewall and disable on system startup:

sudo ufw disable

```
sergey@Server1:~$ sudo ufw status
Status: inactive
sergey@Server1:~$ sudo ufw default allow outgoing
[sudo] password for sergey:
Default outgoing policy changed to 'allow'
(be sure to update your rules accordingly)
sergey@Server1:~$ sudo ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
sergey@Server1:~$ sudo ufw allow ssh
Rules updated
Rules updated (v6)
sergey@Server1:~$ sudo ufw enable
Firewall is active and enabled on system startup
sergey@Server1:~$ sudo ufw status
Status: active
```

To	Action	From
--	-----	----
22/tcp	ALLOW	Anywhere
22/tcp (v6)	ALLOW	Anywhere (v6)

```
sergey@Server1:~$ sudo ufw disable
Firewall stopped and disabled on system startup
sergey@Server1:~$ sudo ufw status
Status: inactive
```

UFW rules

General scheme of rules:

`sudo ufw allow/deny proto tcp/udp from src_addr/prefix port src_port
to dst_addr/prefix port dst_port`

- To open some ports or port ranges:

`sudo ufw allow <port_number>/<tcp or udp>`

Sample: *`sudo ufw allow 80/tcp`*

`sudo ufw allow <start_port_number>:<end_port_number>/<tcp or udp>`

Sample: *`sudo ufw allow 5000:7000/tcp`*

- To allow ALL connections from an IP address or IP network:

`sudo ufw allow from <src_addr/prefix>`

Sample: *`sudo ufw allow from 101.42.20.0/24`*

- Allow access from fixed source udp port and IP address to fixed destination udp port and IP address:

Sample: *`ufw allow proto udp from 10.10.10.10 port 111 to 20.20.20.20 port 222`*

UFW rules editing

- By default, each new rule is added at the end of list
- To delete rule: `sudo ufw delete <rule_number>`
- To insert rule: `sudo ufw insert <rule_number> <rule>`

```
sergey@Server1:~$ sudo ufw status numbered
Status: active

      To      Action      From
      --      -
[ 1] Anywhere  DENY IN    10.0.1.1
[ 2] Anywhere  ALLOW IN   10.0.1.13
[ 3] Anywhere  DENY IN    10.0.1.0/24
[ 4] 22/tcp    ALLOW IN   10.0.1.13
[ 5] 10.0.1.1 23/tcp    DENY IN    10.0.1.13

sergey@Server1:~$ sudo ufw delete 3
Deleting:
 deny from 10.0.1.0/24
Proceed with operation (y|n)? y
Rule deleted
sergey@Server1:~$ sudo ufw status numbered
Status: active

      To      Action      From
      --      -
[ 1] Anywhere  DENY IN    10.0.1.1
[ 2] Anywhere  ALLOW IN   10.0.1.13
[ 3] 22/tcp    ALLOW IN   10.0.1.13
[ 4] 10.0.1.1 23/tcp    DENY IN    10.0.1.13
```

```
sergey@Server1:~$ sudo ufw status numbered
Status: active

      To      Action      From
      --      -
[ 1] Anywhere  DENY IN    10.0.1.1
[ 2] Anywhere  ALLOW IN   10.0.1.13
[ 3] 22/tcp    ALLOW IN   10.0.1.13
[ 4] 10.0.1.1 23/tcp    DENY IN    10.0.1.13

sergey@Server1:~$ sudo ufw insert 3 allow proto udp from 10.0.1.13 to 10.0.1.1
Rule inserted
sergey@Server1:~$ sudo ufw status numbered
Status: active

      To      Action      From
      --      -
[ 1] Anywhere  DENY IN    10.0.1.1
[ 2] Anywhere  ALLOW IN   10.0.1.13
[ 3] 10.0.1.1/udp  ALLOW IN   10.0.1.13/udp
[ 4] 22/tcp    ALLOW IN   10.0.1.13
[ 5] 10.0.1.1 23/tcp    DENY IN    10.0.1.13
```


View the firewall logs

By default all UFW entries are logged into </var/log/ufw.log> file

```
sergey@Server1:~$ sudo more /var/log/ufw.log
Apr 13 16:23:23 Server1 kernel: [26085.302221] [UFW BLOCK] IN=enp0s8 OUT=enp0s3
MAC=08:00:27:ec:6a:64:08:00:27:7c:97:87:08:00 SRC=10.0.1.13 DST=8.8.8.8 LEN=60
TOS=0x00 PREC=0x00 TTL=63 ID=30376 DF PROTO=UDP SPT=42638 DPT=53 LEN=40
Apr 13 16:23:43 Server1 kernel: [26105.488868] [UFW BLOCK] IN=enp0s9 OUT=enp0s3
MAC=08:00:27:1f:20:d2:08:00:27:bc:3b:5e:08:00 SRC=10.0.3.12 DST=8.8.8.8 LEN=60
TOS=0x00 PREC=0x00 TTL=63 ID=7945 DF PROTO=TCP SPT=60638 DPT=53 WINDOW=64240 R
ES=0x00 SYN URGP=0
Apr 13 16:23:51 Server1 kernel: [26113.621187] [UFW BLOCK] IN=enp0s3 OUT= MAC=0
1:00:5e:00:00:fb:06:16:bb:90:06:46:08:00 SRC=192.168.68.107 DST=224.0.0.251 LEN
=32 TOS=0x00 PREC=0x00 TTL=1 ID=24500 PROTO=2
Apr 13 16:23:58 Server1 kernel: [26120.471378] [UFW BLOCK] IN=enp0s3 OUT= MAC=0
1:00:5e:00:00:fb:0e:ed:fe:89:2c:d3:08:00 SRC=192.168.68.109 DST=224.0.0.251 LEN
=32 TOS=0x00 PREC=0x00 TTL=1 ID=39152 PROTO=2
Apr 13 16:23:59 Server1 kernel: [26121.627656] [UFW BLOCK] IN=enp0s3 OUT= MAC=0
1:00:5e:00:00:fb:0a:43:8a:99:ee:91:08:00 SRC=192.168.68.106 DST=224.0.0.251 LEN
=32 TOS=0x00 PREC=0x00 TTL=1 ID=4137 PROTO=2
```

UFW rules reset

To delete rule:

sudo ufw reset

```
sergey@Server1:~$ sudo ufw status numbered
Status: active

      To Action      From
      --
[ 1] Anywhere DENY IN 10.0.1.1
[ 2] Anywhere ALLOW IN 10.0.1.13
[ 3] 10.0.1.1/udp ALLOW IN 10.0.1.13/udp
[ 4] 22/tcp ALLOW IN 10.0.1.13
[ 5] 10.0.1.1 23/tcp DENY IN 10.0.1.13

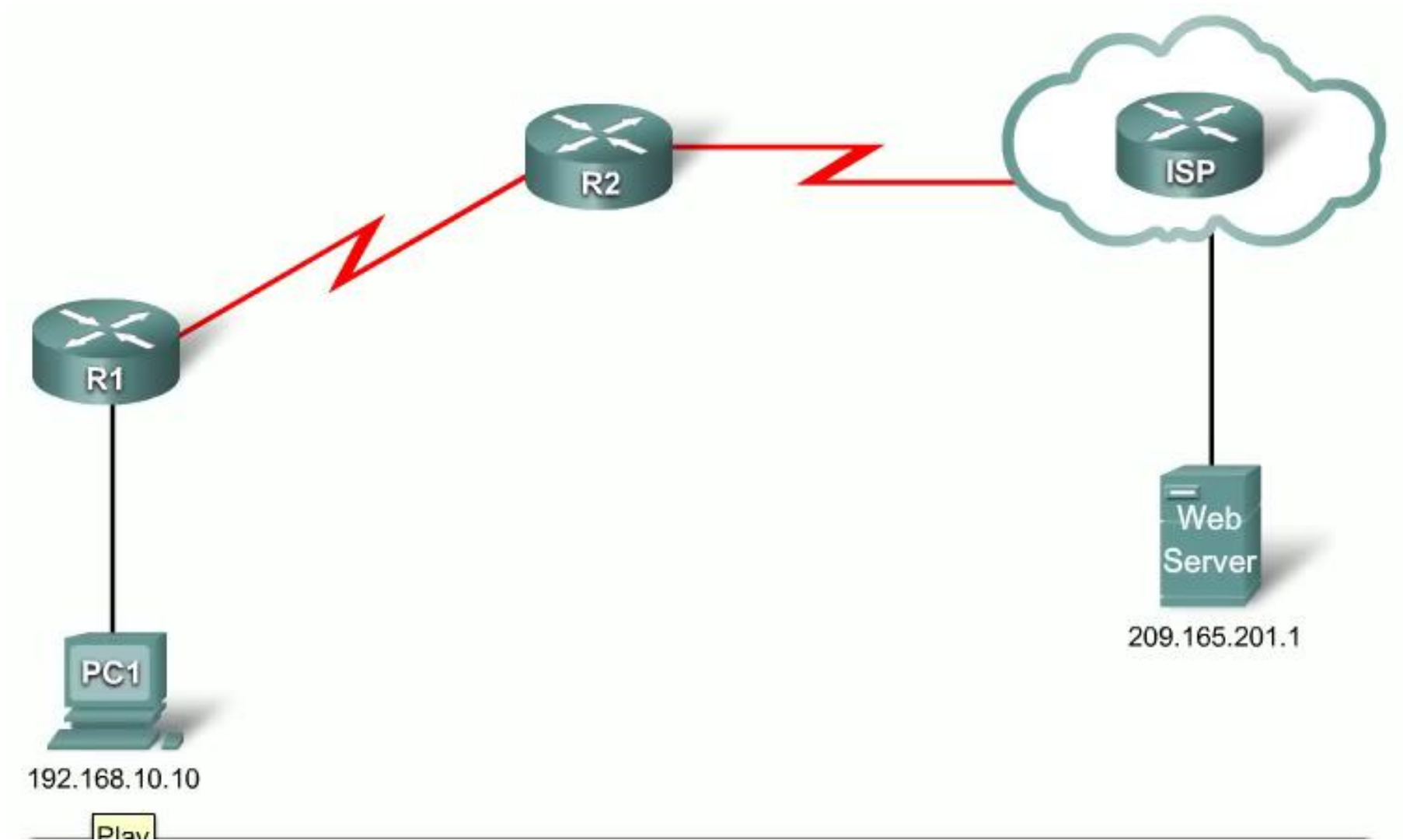
sergey@Server1:~$ sudo ufw reset
Resetting all rules to installed defaults. Proceed with operation (y|n)? y
Backing up 'user.rules' to '/etc/ufw/user.rules.20220413_171412'
Backing up 'before.rules' to '/etc/ufw/before.rules.20220413_171412'
Backing up 'after.rules' to '/etc/ufw/after.rules.20220413_171412'
Backing up 'user6.rules' to '/etc/ufw/user6.rules.20220413_171412'
Backing up 'before6.rules' to '/etc/ufw/before6.rules.20220413_171412'
Backing up 'after6.rules' to '/etc/ufw/after6.rules.20220413_171412'

sergey@Server1:~$ sudo ufw status
Status: inactive
```

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.
- When traffic must be sent/received to/from other organizations or the Internet, the border router translates the addresses to a public and globally unique address

What is NAT?



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

PAT Configuration

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

```
iptables iptables -t nat -A POSTROUTING -j MASQUERADE
```

Sample:

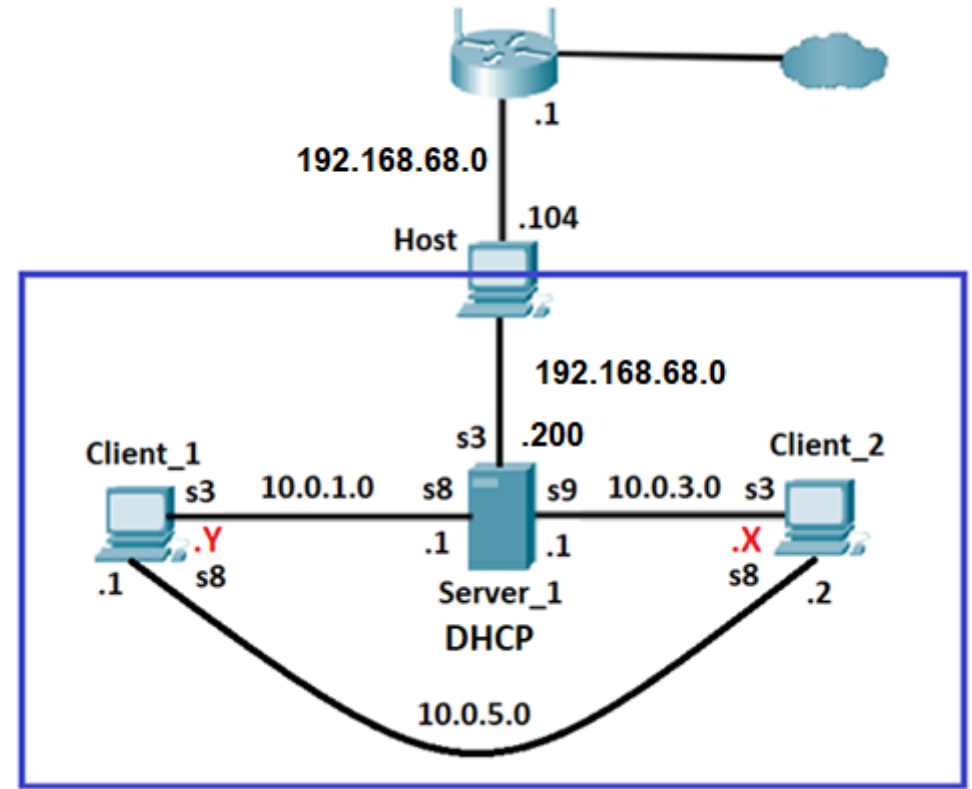
```
iptables -t nat -A POSTROUTING -s 10.0.0.0/16 -j SNAT -  
-to-source 192.168.68.200
```

```
sergey@Server1:~$ sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/16 -j SNAT --
to-source 192.168.68.200
sergey@Server1:~$ sudo iptables -t nat -L
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
SNAT        all  --  10.0.0.0/16           anywhere               to:192.168.68.200
```



```
sergey@Client1:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7171ms

sergey@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=118 time=22.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=23.7 ms
```

Port Forwarding Configuration

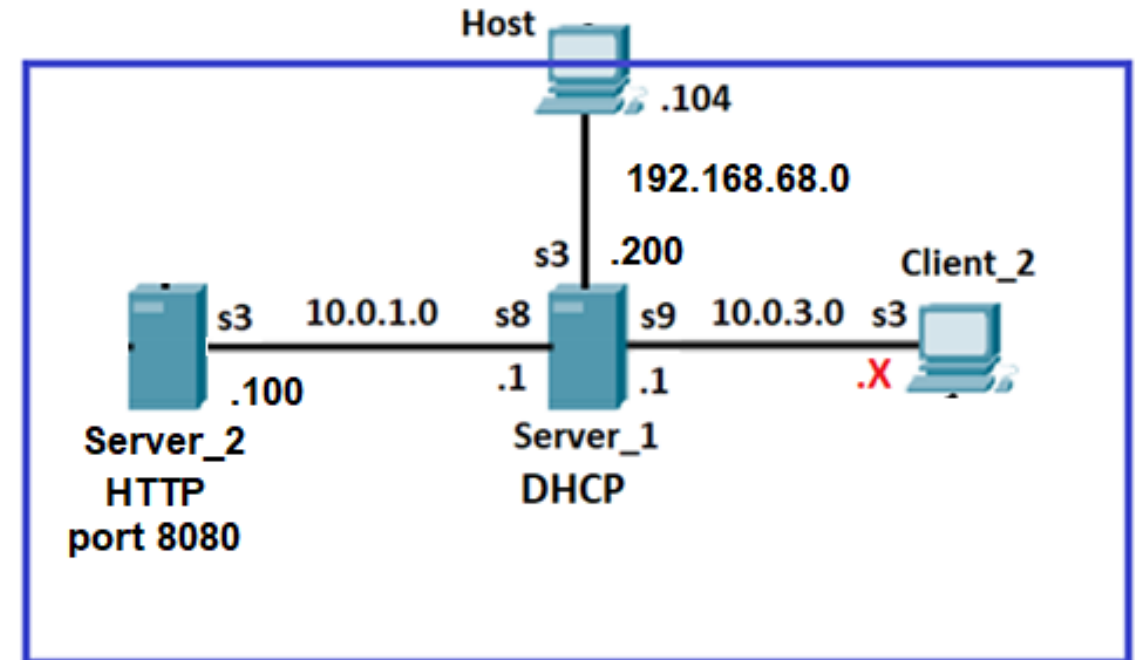
These two rules are needed:

- The first one 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 192.168.1.2.

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



Useful links

- <https://help.ubuntu.com/community/IptablesHowTo>
- <https://upcloud.com/community/tutorials/configure-iptables-ubuntu/>
- <https://linux.die.net/man/8/iptables>
- <https://homes.di.unimi.it/sisop/qemu/iptables-tutorial.pdf>
- [https://www.cyberciti.biz/faq/how-to-configure-firewall-with-ufw-on-ubuntu-20-04-lts/#Set up ufw policy](https://www.cyberciti.biz/faq/how-to-configure-firewall-with-ufw-on-ubuntu-20-04-lts/#Set_up_ufw_policy)
- https://www.howtoforge.com/nat_iptables
- <https://ixnfo.com/nastroyka-maskaradinga-v-ubuntu.html>
- <https://linuxhint.com/configure-nat-on-ubuntu/>
- <http://manpages.ubuntu.com/manpages/xenial/en/man8/ufw.8.html>

Q&A

A light blue world map is centered on the Atlantic Ocean, showing the continents of North America, South America, Europe, Africa, Asia, and Australia. The map is rendered in a simple, stylized manner with thin lines for coastlines and country borders.

Thank you!