**CONFIGURE A FIREWALL IN LINUX**

**Motivation:** The motivation behind configuring a Linux firewall is primarily to bolster security, enforce access control, comply with regulatory standards, segment networks for added protection, monitor network traffic, mitigate attacks, and leverage the flexibility of Linux-based firewall solutions for optimal network protection.

**What is firewall?**

A firewall is a network security device or software application that monitors and controls incoming and outgoing network traffic based on predetermined security rules. its primary purpose is to establish a barrier between a trusted internal network (such as a company's intranet) and untrusted external networks (like the internet), thereby regulating the flow of data packets to prevent unauthorized access and attacks

## **What is UFW?**

UFW, or *uncomplicated firewall*, is a frontend for managing firewall rules in Arch Linux, Debian, or Ubuntu. UFW is used through the command line (although it has GUIs available), and aims to make firewall configuration easy (or, uncomplicated).

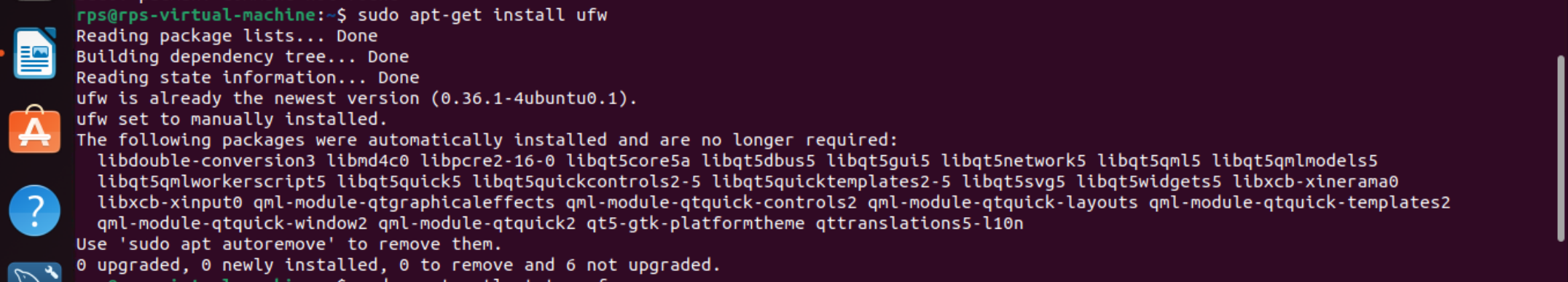
## **Install UFW**

Debian starts UFW’s systemd unit automatically and enables it to start on reboots, but Arch does not. *This is not the same as telling UFW to enable the firewall rules*. Enabling UFW with systemd or upstart only tells the init system to switch on the UFW daemon.

### **Debian / Ubuntu**

Install UFW

sudo apt-get install ufw

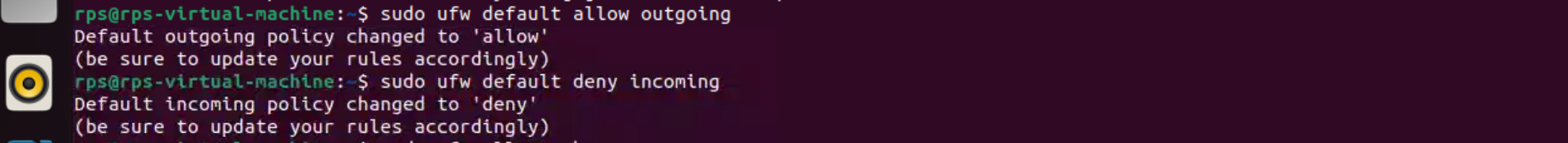


## **Use UFW to Manage Firewall Rules**

### **Set Default Rules**

Most systems need a only a small number of ports open for incoming connections, and all remaining ports closed. To start with an easy basis of rules, the ufw default command can be used to set the default response to incoming and outgoing connections. To deny all incoming and allow all outgoing connections, run:

sudo ufw default allow outgoing  
sudo ufw default deny incoming



The ufw default command also allows for the use of the reject parameter.

**Important**

Configuring a default reject or deny rule can lock you out of your Linode unless explicit allow rules are in place. Ensure that you have configured allow rules for SSH and other critical services as per the section below before applying default deny or reject rules.

### **Add Rules**

Rules can be added in two ways: By denoting the **port number** or by using the **service name**.

For example, to allow both incoming and outgoing connections on port 22 for SSH, you can run:

sudo ufw allow ssh

You can also run:

sudo ufw allow 22

Similarly, to **deny** traffic on a certain port (in this example, 111) you would only have to run:

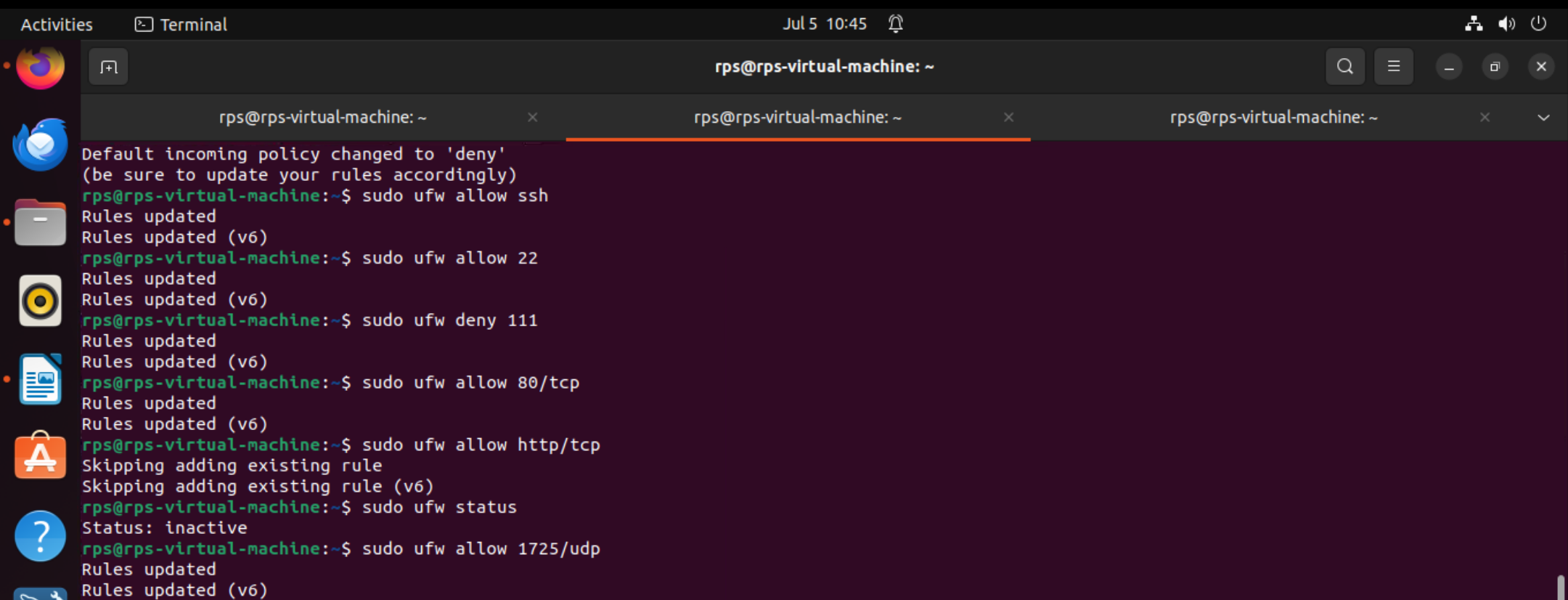
sudo ufw deny 111

To further fine-tune your rules, you can also allow packets based on TCP or UDP. The following allows TCP packets on port 80:

sudo ufw allow 80/tcp  
sudo ufw allow http/tcp

Whereas this will allow UDP packets on 1725:

sudo ufw allow 1725/udp



### **Advanced Rules**

Along with allowing or denying based solely on port, UFW also allows you to allow/block by IP addresses, subnets, and a IP address/subnet/port combinations.

To allow connections from an IP address:

sudo ufw allow from 198.51.100.0

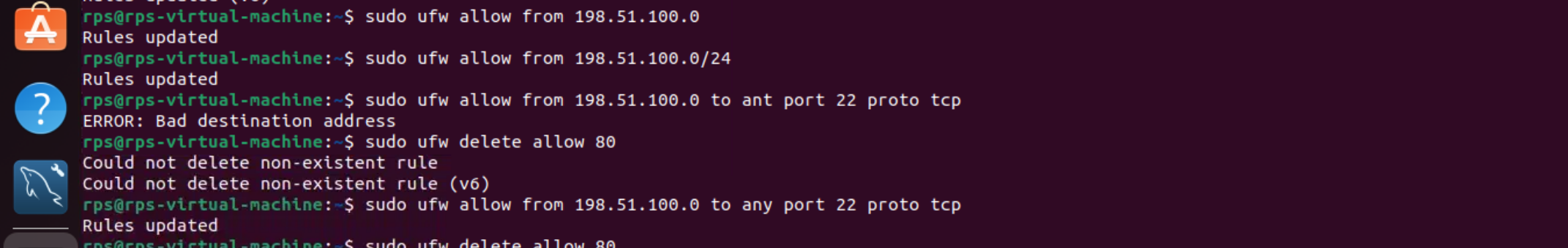
To allow connections from a specific subnet:

sudo ufw allow from 198.51.100.0/24

To allow a specific IP address/port combination:

sudo ufw allow from 198.51.100.0 to any port 22 proto tcp

proto tcp can be removed or switched to proto udp depending upon your needs, and all instances of allow can be changed to deny as needed.



### **Remove Rules**

To remove a rule, add delete before the rule implementation. If you no longer wished to allow HTTP traffic, you could run:

sudo ufw delete allow 80

Deleting also allows the use of service names.

## **Edit UFW’s Configuration Files**

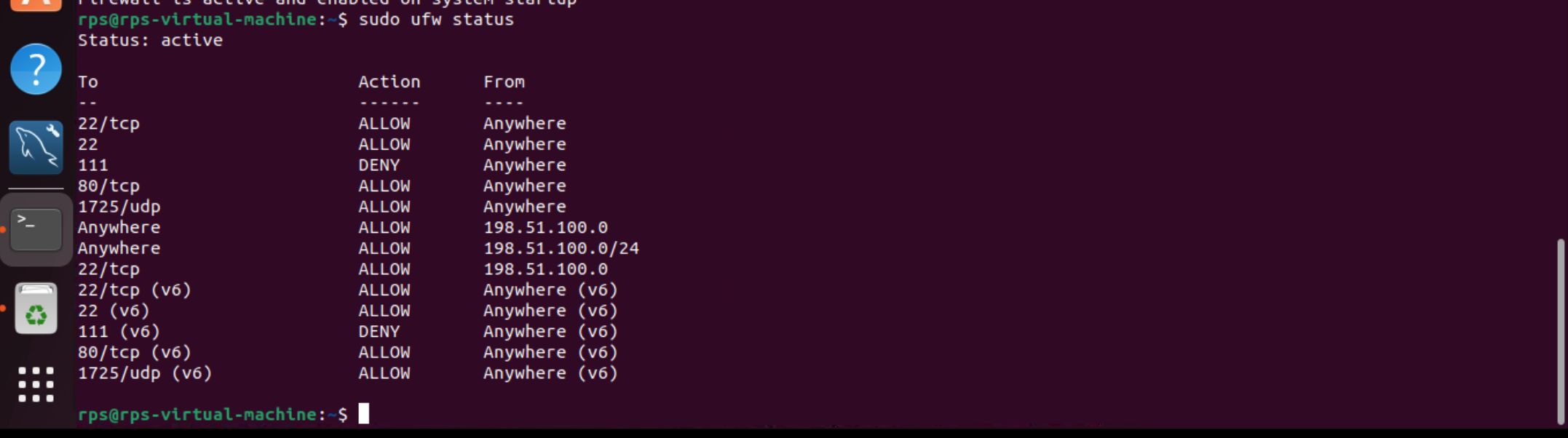
Although simple rules can be added through the command line, there may be a time when more advanced or specific rules need to be added or removed. Prior to running the rules input through the terminal, UFW will run a file, before.rules, that allows loopback, ping, and DHCP. To add to alter these rules edit the /etc/ufw/before.rules file. A before6.rules file is also located in the same directory for IPv6.

An after.rule and an after6.rule file also exists to add any rules that would need to be added after UFW runs your command-line-added rules.

An additional configuration file is located at /etc/default/ufw. From here IPv6 can be disabled or enabled, default rules can be set, and UFW can be set to manage built-in firewall chains.

## **UFW Status**

You can check the status of UFW at any time with the command: sudo ufw status. This will show a list of all rules, and whether or not UFW is active:



### **Enable the Firewall**

With your chosen rules in place, your initial run of ufw status will probably output Status: inactive. To enable UFW and enforce your firewall rules:

sudo ufw enable

Similarly, to disable UFW’s rules:

sudo ufw disable

**Note**

This still leaves the UFW service running and enabled on reboots.

## **Logging**

You can enable logging with the command:

sudo ufw logging on

Log levels can be set by running sudo ufw logging low|medium|high, selecting either low, medium, or high from the list. The default setting is low.

Sep 16 15:08:14 <hostname> kernel: [UFW BLOCK] IN=eth0 OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:00:00 SRC=123.45.67.89 DST=987.65.43.21 LEN=40 TOS=0x00 PREC=0x00 TTL=249 ID=8475 PROTO=TCP SPT=48247 DPT=22 WINDOW=1024 RES=0x00 SYN URGP=0

## **About iptables**

iptables is a command-line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on your system, iptables looks for a rule in its list to match it to. If it doesn't find one, it resorts to the default action.

iptables almost always comes pre-installed on any Linux distribution. To update/install it, just retrieve the iptables package:

sudo apt-**get** install iptables

There are GUI alternatives to iptables like [Firestarter](https://sourceforge.net/projects/firestarter/), but iptables isn't really that hard once you have a few commands down. You want to be extremely careful when configuring iptables rules, particularly if you're SSH'd into a server, because one wrong command can permanently lock you out until it's manually fixed at the physical machine. And don't forget to [lock down your SSH server](https://www.howtogeek.com/devops/how-to-lock-down-your-ssh-server/) if you open the port.

## **Types of Chains**

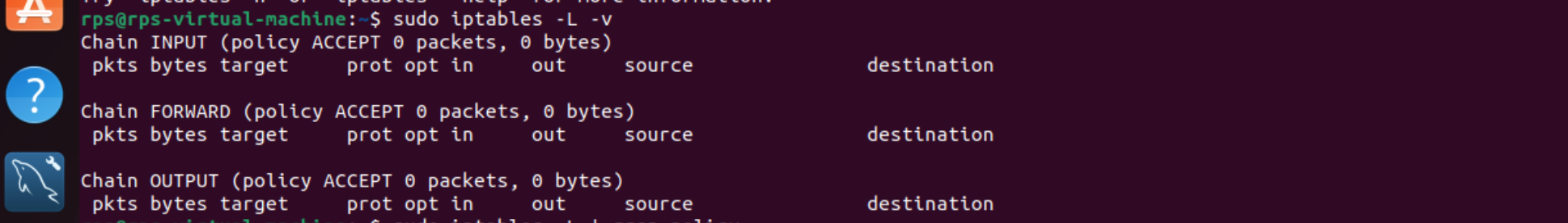
iptables uses three different chains: input, forward, and output.

**Input** - This chain is used to control the behavior for incoming connections. For example, if a user attempts to SSH into your PC/server, iptables will attempt to match the IP address and port to a rule in the input chain.

**Forward** - This chain is used for incoming connections that aren't actually being delivered locally. Think of a router - data is always being sent to it but rarely actually destined for the router itself; the data is just forwarded to its target. Unless you're doing some kind of routing, NATing, or something else on your system that requires forwarding, you won't even use this chain.

There's one sure-fire way to check whether or not your system uses/needs the forward chain.

iptables -L -v



The screenshot above is of a server that's been running for a few weeks and has no restrictions on incoming or outgoing connections. As you can see, the input chain has processed 11GB of packets and the output chain has processed 17GB. The forward chain, on the other hand, has not needed to process a single packet. This is because the server isn't doing any kind of forwarding or being used as a pass-through device.

**Output** - This chain is used for outgoing connections. For example, if you try to ping howtogeek.com, iptables will check its output chain to see what the rules are regarding ping and howtogeek.com before making a decision to allow or deny the connection attempt.

**The caveat**

Even though pinging an external host seems like something that would only need to traverse the output chain, keep in mind that to return the data, the input chain will be used as well. When using iptables to lock down your system, remember that a lot of protocols will require two-way communication, so both the input and output chains will need to be configured properly. SSH is a common protocol that people forget to allow on both chains.

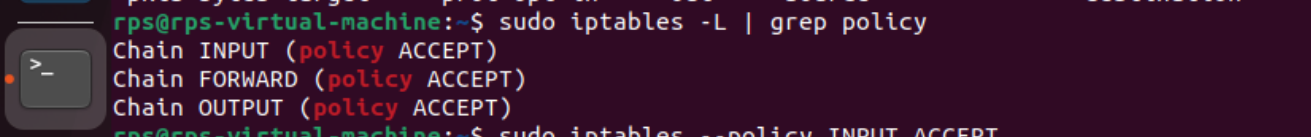
## **Policy Chain Default Behavior**

Before going in and configuring specific rules, you'll want to decide what you want the default behavior of the three chains to be. In other words, what do you want iptables to do if the connection doesn't match any existing rules?

To see what your policy chains are currently configured to do with unmatched traffic, run the

iptables -L

command.



As you can see, we also used the grep command to give us cleaner output. In that screenshot, our chains are currently figured to accept traffic.

More times than not, you'll want your system to accept connections by default. Unless you've changed the policy chain rules previously, this setting should already be configured. Either way, here's the command to accept connections by default:

iptables --policy INPUT ACCEPT  
iptables --policy OUTPUT ACCEPT  
iptables --policy FORWARD ACCEPT

By defaulting to the accept rule, you can then use iptables to deny specific IP addresses or port numbers, while continuing to accept all other connections. We'll get to those commands in a minute.

If you would rather deny all connections and manually specify which ones you want to allow to connect, you should change the default policy of your chains to drop. Doing this would probably only be useful for servers that contain sensitive information and only ever have the same IP addresses connect to them.

iptables --policy INPUT DROP  
iptables --policy OUTPUT DROP  
iptables --policy FORWARD DROP

## **Connection-specific Responses**

With your default chain policies configured, you can start adding rules to iptables so it knows what to do when it encounters a connection from or to a particular IP address or port. In this guide, we're going to go over the three most basic and commonly used "responses".

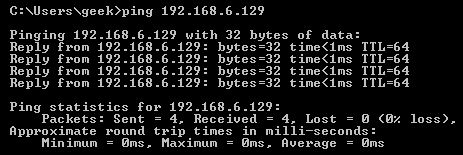
**Accept** - Allow the connection.

**Drop** - Drop the connection, act like it never happened. This is best if you don't want the source to realize your system exists.

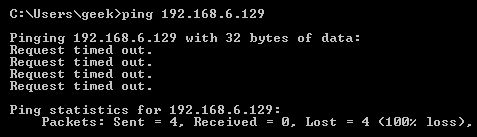
**Reject** - Don't allow the connection, but send back an error. This is best if you don't want a particular source to connect to your system, but you want them to know that your firewall blocked them.

The best way to show the difference between these three rules is to show what it looks like when a PC tries to ping a Linux machine with iptables configured for each one of these settings.

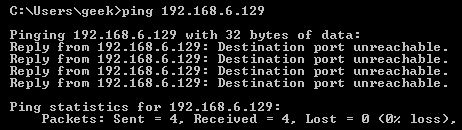
Allowing the connection:



Dropping the connection:



Rejecting the connection:



## **Allowing or Blocking Specific Connections**

With your policy chains configured, you can now configure iptables to allow or block specific addresses, address ranges, and ports. In these examples, we'll set the connections to

**DROP**

, but you can switch them to

ACCEPT

or

REJECT

, depending on your needs and how you configured your policy chains.

Note: In these examples, we're going to use

iptables -A

to append rules to the existing chain. iptables starts at the top of its list and goes through each rule until it finds one that it matches. If you need to insert a rule above another, you can use

**iptables** **-I** [chain] [number]

to specify the number it should be in the list.

**Connections from a single IP address**

This example shows how to block all connections from the IP address 10.10.10.10.

**iptables** **-A** **INPUT** **-s** 10.10.10.10 **-j** **DROP**

**Connections from a range of IP addresses**

This example shows how to block all of the IP addresses in the 10.10.10.0/24 network range. You can use a netmask or standard slash notation to specify the range of IP addresses.

iptables -A INPUT -s 10.10.10.0/24 -j **DROP**

or

iptables -A INPUT -s 10.10.10.0/255.255.255.0 -j **DROP**

**Connections to a specific port**

This example shows how to block SSH connections from 10.10.10.10.

**iptables** **-A** **INPUT** **-p** **tcp** **--dport** **ssh** **-s** 10.10.10.10 **-j** **DROP**

You can replace "ssh" with any protocol or port number. The

-p tcp

part of the code tells iptables what kind of connection the protocol uses. If you were blocking a protocol that uses UDP rather than TCP, then

-p udp

would be necessary instead.

This example shows how to block SSH connections from any IP address.

iptables -A INPUT -p tcp --dport ssh -j DROP

## **Connection States**

As we mentioned earlier, a lot of protocols are going to require two-way communication. For example, if you want to allow SSH connections to your system, the input and output chains are going to need a rule added to them. But, what if you only want SSH coming into your system to be allowed? Won't adding a rule to the output chain also allow outgoing SSH attempts?

That's where connection states come in, which give you the capability you'd need to allow two way communication but only allow one way connections to be established. Take a look at this example, where SSH connections FROM 10.10.10.10 are permitted, but SSH connections TO 10.10.10.10 are not. However, the system is permitted to send back information over SSH as long as the session has already been established, which makes SSH communication possible between these two hosts.

**iptables** **-A** **INPUT** **-p** **tcp** **--dport** **ssh** **-s** 10.10.10.10 **-m** **state** **--state** **NEW**,**ESTABLISHED** **-j** **ACCEPT**  
**iptables** **-A** **OUTPUT** **-p** **tcp** **--sport** 22 **-d** 10.10.10.10 **-m** **state** **--state** **ESTABLISHED** **-j** **ACCEPT**

## **Saving Changes**

The changes that you make to your iptables rules will be scrapped the next time that the iptables service gets restarted unless you execute a command to save the changes. This command can differ depending on your distribution:

**Ubuntu:**

sudo /sbin/iptables-save

**Red Hat / CentOS:**

/sbin/service iptables save

**Or**

/etc/init.d/iptables save

## **Other Commands**

List the currently configured iptables rules:

iptables -L

Adding the

-v

option will give you packet and byte information, and adding

-n

will list everything numerically. In other words - hostnames, protocols, and networks are listed as numbers.

To clear all the currently configured rules, you can issue the flush command.

iptables -F

**Conclusion**

In conclusion, configuring UFW in Linux not only enhances network security but also simplifies the management of firewall rules, promotes compliance with industry standards, and facilitates effective monitoring and access control over network traffic.