March 22, 2019 18 min read

How to setup your own VPN server using WireGuard on Ubuntu

Perhaps you've heard of WireGuard - the new VPN protocol that utilizes state-of-the-art cryptography. It's super fast, extremely simple and considerably more performant than OpenVPN. A VPN will protect you against Man-in-the-Middle attacks (especially if you're using public WiFi networks, even if they are password protected), guard your privacy against ISPs that are snooping into your traffic and selling your data, and will help prevent censorship in countries where digital freedom is restricted.

Personally, I've been blown away by its features! My jaw dropped when I saw that it establishes the connection in less than 100ms. WireGuard works over UDP (by default on port 51820) has a very simple handshake that occurs every few minutes in order to ensure perfect forward secrecy. It has IP roaming support so

you can simply disconnect from a WiFi and connect to another and it will simply work. Imagine this: I put my laptop in sleep mode when I leave my office and when I open it at home my ssh sessions are still alive and I can use them right away! The codebase itself is very clean and Linus himself expressed his willingness to see the WireGuard in the Linux kernel soon. The authentication between peers works using Curve25519 key pairs for ECDH.

It is worth to mention that in WireGuard's terms there is no "server" and "client" - each device that is connected is rather a "peer". In this tutorial though we will use the "server" term to refer to the VPS that will be used to route traffic through and the "client" term for the devices connecting to the server for this purpose.

Enough with the theory, let's go ahead and setup your first WireGuard VPN server!

Get a VPS

Heads up: This tutorial was tested using a Ubuntu 18.04 server, although it should be very similar for other versions or Linux distributions.

First of all, you'll need a VPS that you will use as a VPN server. WireGuard is very lightweight so the cheapest VPS that has a public IPv4 will be probably more than enough.

Tip: If you don't have a VPS provider you can sign up for an account at Vultr. By using this referral link you get free 50$ which practically gives you a 5$ VPS for 10 months for free at the time of writing this article.

After signing up create an Ubuntu 18.04 VPS in a region of your preference. If you don't know which to choose - pick one that's geographically closer to you. You can tick the IPv6 checkbox if you want to be able to access IPv6 through the VPN as well.

Installation

Once your VPS is ready you need to ssh into it. On Linux and macOS you can use the type `ssh root@your-server-ip-address` to connect to your server. On

Windows, you will need an SSH client such as Putty. All the commands in this tutorial should be executed as `root` so you have to execute `sudo su` to become root beforehand if you're using a different user.

WireGuard has its own PPA repository which we'll have to add in order to install the latest version and keep it up to date.

```
add-apt-repository ppa:wireguard/wireguard
apt-get update # you can skip this on Ubuntu 18.04
apt-get install wireguard
```

WireGuard works as a kernel module that is installed using DKMS so every time you upgrade your kernel - the WireGuard kernel module is automatically compiled and ready to use for your new kernel as well. In order to use the kernel module right after the installation you have to either reboot or run `modprobe` to activate it:

```
modprobe wireguard
```

You can check whether the kernel module is loaded using `lsmod | grep wireguard`. If the module is properly loaded you should see an output similar to this:

```
root@wg-server:~# lsmod | grep wireguard
wireguard              208896  0
ip6_udp_tunnel          16384  1 wireguard
udp_tunnel              16384  1 wireguard
root@wg-server:~#
```

Now you're ready to get to the next step.

Generating keys

WireGuard comes with two useful command-line utilities: `wg` and `wg-quick`. `wg` is the configuration utility for getting and setting the configuration of

WireGuard tunnel interfaces. `wg-quick` on the other hand is a simple script for easily bringing up a WireGuard interface.

To generate the public and private keys, use the following commands:

```
cd /etc/wireguard
umask 077
wg genkey | sudo tee privatekey | wg pubkey | sudo tee publickey
```

This will create two files: `publickey` and `privatekey` that will contain the public and private key respectively.

Pro tip: You can use wireguard-vanity-address to generate a *pretty* public key, for example, that starts with certain characters. It's easier to identify your mobile phone's connection in the peer list if its public key starts with "pho" :). Keep in mind that you will need a safe transport in order to move the private key to the device. Generally, it is not recommended to have the private key on a different device than the one that will use it.

You will need to generate a key-pair for every peer (device) that is connected. So for example, if you have a laptop and a mobile phone - you'll need 3 key-pairs. One pair on the server, one on the laptop, and one on the mobile phone. Depending on the WireGuard client that you are using you will probably be able to generate the keys on the device itself.

Configuration

Next, we'll need to create the configuration for the `wg0` device that will route our VPN traffic. It doesn't have to be named `wg0` - you can name the device however you like, `wg0` is just merely a convention.

You will have to choose an IP subnet to be used for your VPN network. In this example, I'm using `10.10.0.1/24`. Make sure you choose a subnet that is not used by your home and/or work networks, otherwise you either won't be able to access devices on the local network or the setup will not work at all.

Using your favorite editor (vim, nano, etc.) create the `/etc/wireguard/wg0.conf` with the following contents:

```
[Interface]
PrivateKey = <your server private key here>
Address = 10.10.0.1/24
Address = fd86:ea04:1111::1/64
SaveConfig = true
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A I
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -I
ListenPort = 51820
```

Heads up: Make sure you replace `<your server private key here>` with your actual private key that you generated in the previous step, `10.10.0.1/24, fd172.21:12::1/64` with the actual IPs that you intend to use (if different), and `ens3` with the name of your server's main network interface (that you connect to the internet with, for example, `eth0`) if it is different from ens3. You can list your network interfaces with the `ifconfig` or `ip a` commands.

The configuration is perhaps self-explanatory except for the `SaveConfig` option. Unlike OpenVPN or other VPN servers, WireGuard does not need a "restart" or "reload" in order for the client to be able to connect - all you need to do is to use the `wg` command-line utility to add your peer and you are able to connect right away. This is an in-memory operation though and if you restart the server - your added peers will be gone. Here's when `SaveConfig` option comes in! This option is used to tell wg-quick to automatically save the newly added peers to your configuration file. This might also bite you - if you edit the file while the interface is up - you will lose your modifications upon restart.

Keep in mind that the `wg-quick` configuration file that we are using is a superset (of sorts) of the configuration file accepted by `wg`. For this reason, you cannot use this config file with `wg` directly but only with `wg-quick`.

Next, we will need to tell the Linux kernel to forward the traffic so our NAT will work. To do this you need to edit the `/etc/sysctl.conf` file and add these two lines and enable the changes using `sysctl -p`:

```
cat << EOF >> /etc/sysctl.conf
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
EOF
sysctl -p
```

Now you can bring up your wg0 interface:

```
wg-quick up wg0
```

If everything went well, you can run `wg` in order to see the currently active interfaces and connected peers. The output should look something like this:

```
root@wg-server:~# wg
interface: wg0
  public key: wYHnSnbRraJNooN5gjNtH5SLS1tw/0xehAVbU9bgIxA=
  private key: (hidden)
  listening port: 51820
```

Client setup

Now that our VPN server is up we can connect our clients to it. In this example, I will be using an Ubuntu 18.04 for the client as well, but the steps are similar for other operating systems. Some GUI clients offer to generate or import the configuration from a file as well.

If you're using a Linux distribution you can use the official installation page to find the commands to install WireGuard. If you are like me, you can use the command-line client. Alternatively, there are some GUI clients available that you can use:

macOS: https://itunes.apple.com/us/app/wireguard/id1451685025
Gnome's Network-Manager: https://github.com/max-moser/network-manager-wireguard
Android: https://play.google.com/store/apps/details?id=com.wireguard.android
iOS: https://itunes.apple.com/us/app/wireguard/id1441195209

There is no official Windows client yet but there are other proprietary options like Tunsafe.

Heads up: The WireGuard developers are strongly advising to stay away from Windows clients that are not released from the official site, as they may be dangerous to use, despite marketing efforts.

Adding peers

The client configuration is similar to the server one with just a few differences. First of all, you must generate the key-pair just like we did for the server. Afterward, you need to setup the network interface, we which we'll call `wg0` as well.

On your **client** device, create the `/etc/wireguard/wg0.conf` file with the following contents:

```
[Interface]
Address = 10.10.0.2/32
Address = fd86:ea04:1111::2/128
SaveConfig = true
PrivateKey = <your client private key here>
DNS = 1.1.1.1

[Peer]
PublicKey = <your server public key here>
Endpoint = <your server public ip>:51820
AllowedIPs = 0.0.0.0/0, ::/0
```

Make sure to replace the client private key, server public key and server IP address respectively. The `AllowedIPs` will be used to determine which traffic to forward through the VPN. You can think of it as a "pull route" in OpenVPN's terms. By setting it to `0.0.0.0/0, ::/0` you are instructing your VPN client to route all traffic through your VPN server.

To make sure your config file is safe, set the right permissions on it:

```
chmod 600 /etc/wireguard/wg0.conf
```

As a next step, you can now add your client public key to your server. Run this command on your **server**:

```
wg set wg0 peer <client-public-key> allowed-ips 10.10.0.2/32,fd86:e
```

Make sure to replace `<client-public-key>` with your actual client public key and adjust the IP addresses for your client if they are different. Next, you can run `wg` to confirm that your peer was successfully added. The output should look similar to this one:

```
root@wg-server:~# wg
interface: wg0
  public key: wYHnSnbRraJNooN5gjNtH5SLS1tw/0xehAVbU9bgIxA=
  private key: (hidden)
  listening port: 51820

peer: m5S0sA0NWlbEU6+5oegudniHgwF07GSKcWs2tt3LZEI=
  allowed ips: 10.10.0.2/32, fd86:ea04:1111::/128
```

You can now activate the `wg0` interface on your **client** device using the `wg-quick up wg0` command. The output should look similar to this:

```
# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip address add 10.10.0.2/32 dev wg0
[#] ip address add fd86:ea04:1111::2/128 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] ip -6 route add ::/0 dev wg0 table 51820
[#] ip -6 rule add not fwmark 51820 table 51820
[#] ip -6 rule add table main suppress_prefixlength 0
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
```

You can also run the `wg` utility and check the output of your active interfaces and connected peers.

If everything went well, you now should be connected to the VPN server and all your traffic should be routed through it. You can check your IP address using ipx.ac. You can also run all the tests in order to make sure that your location is not leaked.

Starting the VPN at system startup

Since Ubuntu is using systemd, you can easily enable WireGuard on system boot:

```
systemctl enable wg-quick@wg0
```

You should run this command on both the server and client. If you're using a GUI client you probably don't need to run this command on the client though.

Firewall port whitelisting

In case you are using a firewall to whitelist the IPs and/or ports that can communicate with your server, you will need to add an exception for the port used by your WireGuard server. First of all, you'll need to determine if you're using a firewall. On Ubuntu, the two most common options are `iptables` and `ufw` (which uses `iptables` behind the scenes anyway).

UFW

UFW stands for "Uncomplicated Firewall". To check whether you're using `ufw` you need to run the `ufw status verbose` command:

```
# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip
```

If the status is reported as being inactive - you can skip this step. If the `ufw` is enabled you need to run the following command in order to whitelist the 51820 port:

```
ufw allow 51820/udp
```

iptables

The second option is iptables which is the de facto firewall for Linux operating systems. To list the iptables rules use the `iptables -L -n` command. The output might look something like this:

```
# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp d
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp d
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp d

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp s
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp s
```

The important parts that you're looking for are:

- The policy (ACCEPT/DROP)
- The iptables rules: in our case for the UDP port 51820

If the policy for the chains is ACCEPT and the port is not rejected by any particular rule - you're probably good to go. However if the default policy is to drop or reject the packets, you need to add new rules in order to allow the connections to our VPN server:

```
iptables -A INPUT -p udp -m udp --dport 51820 -j ACCEPT
iptables -A OUTPUT -p udp -m udp --sport 51820 -j ACCEPT
```

Conclusion

I hope this tutorial is a good start towards better security for yourself. In case you're having any troubles setting it up you can write us on Twitter @secespresso or join our Telegram group secespresso so our community can give you some tips. Don't expect anyone to set it up for you - it is not a support forum but rather a community effort to help each other. If you spotted any typos, errors, problems with this article or you just want to add some more information to it - you are welcome to check our GitHub and make a pull request with the required change.

WireGuard is a great VPN protocol that is believed to be the VPN protocol of the future. Obviously, there are commercial alternatives available (for ex.: vpn.ac, mullvad.net, azirevpn.com) in case you are struggling setting it up yourself, but using a VPN is always a good idea.