Set Up Your Own WireGuard VPN Server on Debian

This tutorial is going to show you how to set up your own WireGuard VPN server on Debian. WireGuard is made specifically for the Linux kernel. It runs inside the Linux kernel and allows you to create fast, modern, and secure VPN tunnel.

Features of WireGuard VPN

- Lightweight and super fast speed, blowing OpenVPN out of the water.
- Cross-platform. WireGuard can run on Linux, BSD, macOS, Windows, Android, iOS, and OpenWRT.
- User authentication is done by exchanging public keys, similar to SSH keys.
- It assigns static tunnel IP addresses to VPN clients. Some folks may not like it, but it can be useful in some cases.
- Mobile devices can switch between Wi-Fi and mobile network seamlessly without dropping any connectivity.
- It aims to replace OpenVPN and IPSec in most use cases.

Prerequisites

This tutorial assumes that the VPN server and VPN client are both running **Debian** operating system.

Step 1: Install WireGuard on Debian Server and Desktop

Log into your Debian server. WireGuard is included in the Debian 11 (Bullseye) repository, so you can run the following commands to install it.

```
sudo apt update
```

```
sudo apt install wireguard wireguard-tools linux-headers-$(uname -r)
```

Debian 10 users need to add the backport repository with the following command.

```
echo "deb http://deb.debian.org/debian buster-backports main" | sudo tee
/etc/apt/sources.list.d/buster-backports.list
```

Then install WireGuard.

```
sudo apt update
sudo apt -t buster-backports install wireguard wireguard-tools wireguard-dkms linux-headers-
$(uname -r)
```

Use the same commands to install WireGuard on your local Debian computer (the VPN client). Note that you also need to install the `openresolv` package on the client to configure DNS server.

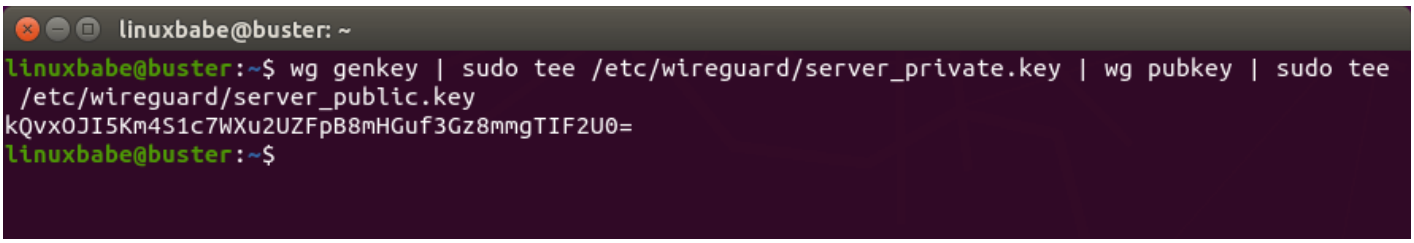
```
sudo apt install openresolv
```

Step 2: Generate Public/Private Keypair

Server

Run the following command on the Debian server to create a public/private key pair, which will be saved under `/etc/wireguard/` directory.

```
wg genkey | sudo tee /etc/wireguard/server_private.key | wg pubkey | sudo tee
/etc/wireguard/server_public.key
```



Client

Run the following command to create a public/private key pair on the local Debian computer (the VPN client).

```
wg genkey | sudo tee /etc/wireguard/client_private.key | wg pubkey | sudo tee /etc/wireguard/client_public.key
```

Step 3: Create WireGuard Configuration File

Server

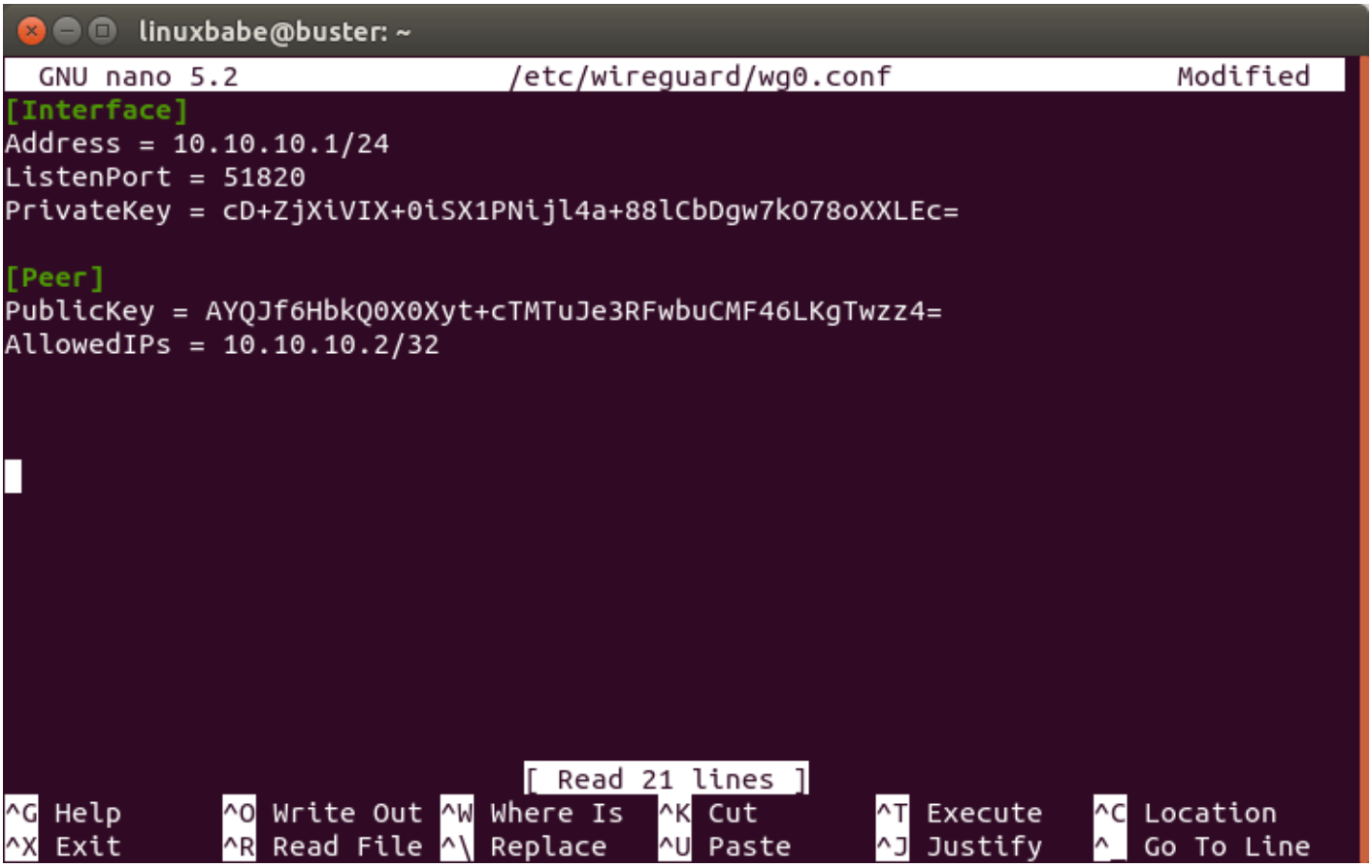
Use a command-line text editor like Nano to create a WireGuard configuration file on the Debian server. `wg0` will be the network interface name.

```
sudo nano /etc/wireguard/wg0.conf
```

Copy the following text and paste it to your configuration file. You need to use your own server private key and client public key.

```
[Interface]
Address = 10.10.10.1/24
ListenPort = 51820
PrivateKey = cD+ZjXiVIX+0iSX1PNijl4a+88lCbDgw7kO78oXXLEc=

[Peer]
PublicKey = AYQJf6HbkQ0X0Xyt+cTMTuJe3RFwbuCMF46LKgTwzz4=
AllowedIPs = 10.10.10.2/32
```



Where:

- **Address**: Specify the private IP address of the VPN server. Here I'm using the 10.10.10.0/24 network range, so it won't conflict with your home network range. (Most home routers use 192.168.0.0/24 or 192.168.1.0/24). 10.10.10.1 is the private IP address for the VPN server.
- **PrivateKey**: The private key of VPN server, which can be found in the `/etc/wireguard/server_private.key` file on the server.
- **ListenPort**: WireGuard VPN server will be listening on UDP port 51820, which is the default.
- **PublicKey**: The public key of VPN client, which can be found in the `/etc/wireguard/client_public.key` file on the client computer.
- **AllowedIPs**: IP addresses the VPN client is allowed to use. In this example, the client can only use the 10.10.10.2 IP address inside the VPN tunnel.

Save and close the file. (To save a file in Nano text editor, press `Ctrl+O`, then press Enter to confirm. Press `Ctrl+X` to exit.)

Change the file permission mode so that only root user can read the files.

```
sudo chmod 600 /etc/wireguard/ -R
```

Client

Use a command-line text editor like Nano to create a WireGuard configuration file on your local

Debian computer. `wg-client0` will be the network interface name.

```
sudo nano /etc/wireguard/wg-client0.conf
```

Copy the following text and paste it to your configuration file. You need to use your own client private key and server public key.

```
[Interface]
Address = 10.10.10.2/24
DNS = 10.10.10.1
PrivateKey = cOFA+x5UvHF+a3xJ6enLatG+DoE3I5PhMgKrMKkUyXI=

[Peer]
PublicKey = kQvxOJI5Km4S1c7WXu2UZFpB8mHGuf3Gz8mmgTIF2U0=
AllowedIPs = 0.0.0.0/0
Endpoint = 12.34.56.78:51820
PersistentKeepalive = 25
```

Where:

- **Address**: Specify the private IP address of the VPN client.
- **DNS**: specify 10.10.10.1 (the VPN server) as the DNS server. It will be configured via the `resolvconf` command. You can also specify multiple DNS servers for redundancy like this: `DNS = 10.10.10.1 8.8.8.8`
- **PrivateKey**: The client's private key, which can be found in the `/etc/wireguard/client_private.key` file on the client computer.
- **PublicKey**: The server's public key, which can be found in the `/etc/wireguard/server_public.key` file on the server.
- **AllowedIPs**: 0.0.0.0/0 represents the whole Internet, which means all traffic to the Internet should be routed via the VPN.
- **Endpoint**: The public IP address and port number of VPN server. Replace 12.34.56.78 with your server's real public IP address.
- **PersistentKeepalive**: Send an authenticated empty packet to the peer every 25 seconds to keep the connection alive. If PersistentKeepalive isn't enabled, the VPN server might not be able to ping the VPN client.

Save and close the file.

Change the file mode so that only root user can read the files.

```
sudo chmod 600 /etc/wireguard/ -R
```

Step 4: Enable IP Forwarding on the Server

In order for the VPN server to route packets between VPN clients and the Internet, we need to enable IP forwarding. Edit `sysctl.conf` file.

```
sudo nano /etc/sysctl.conf
```

Add the following line at the end of this file.

```
net.ipv4.ip_forward = 1
```

Save and close the file. Then apply the changes with the below command. The **-p** option will load sysctl settings from **/etc/sysctl.conf** file. This command will preserve our changes across system reboots.

```
sudo sysctl -p
```

Step 5: Configure IP Masquerading on the Server

We need to set up IP masquerading in the server firewall, so that the server becomes a virtual router for VPN clients. I will use UFW, which is a front end to the iptables firewall. Install UFW on Debian with:

```
sudo apt install ufw
```

First, you need to allow SSH traffic.

```
sudo ufw allow 22/tcp
```

Next, find the name of your server’s main network interface.

```
ip addr
```

As you can see, it's named `ens3` on my Debian server.

```
linuxbabe@buster: ~
linuxbabe@buster:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3  <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 56:00:02:2b:d4:38 brd ff:ff:ff:ff:ff:ff
    inet 155.138.129.48/23 brd 155.138.129.255 scope global dynamic ens3
       valid_lft 82565sec preferred_lft 82565sec
    inet6 2001:19f0:b001:374:5400:2ff:fe2b:d438/64 scope global dynamic mngtmpaddr
       valid_lft 2591900sec preferred_lft 604700sec
    inet6 fe80::5400:2ff:fe2b:d438/64 scope link
       valid_lft forever preferred_lft forever
linuxbabe@buster:~$
```

To configure IP masquerading, we have to add iptables command in a UFW configuration file.

```
sudo nano /etc/ufw/before.rules
```

By default, there are some rules for the `filter` table. Add the following lines at the end of this file. Replace `ens3` with your own network interface name.

```
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o ens3 -j MASQUERADE


# End each table with the 'COMMIT' line or these rules won't be processed
COMMIT
```

In Nano text editor, you can go to the end of the file by pressing `Ctrl+W`, then pressing `Ctrl+V`.

```
# allow MULTICAST UPnP for service discovery (be sure the MULTICAST line above
# is uncommented)
-A ufw-before-input -p udp -d 239.255.255.250 --dport 1900 -j ACCEPT

# don't delete the 'COMMIT' line or these rules won't be processed
COMMIT

# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o ens3 -j MASQUERADE

# End each table with the 'COMMIT' line or these rules won't be processed
COMMIT
```

The above lines will append (**-A**) a rule to the end of of **POSTROUTING** chain of **nat** table. It will link your virtual private network with the Internet. And also hide your network from the outside world. So the Internet can only see your VPN server's IP, but can't see your VPN client's IP, just like your home router hides your private home network.

By default, UFW forbids packet forwarding. We can allow forwarding for our private network. Find the `ufw-before-forward` chain in this file and add the following 3 lines, which will accept packet forwarding if the source IP or destination IP is in the `10.10.10.0/24` range.

```
# allow forwarding for trusted network
-A ufw-before-forward -s 10.10.10.0/24 -j ACCEPT
-A ufw-before-forward -d 10.10.10.0/24 -j ACCEPT
```

```
# ok icmp code for FORWARD
-A ufw-before-forward -p icmp --icmp-type destination-unreachable -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type time-exceeded -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type parameter-problem -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type echo-request -j ACCEPT

# allow forwarding for trusted network
-A ufw-before-forward -s 10.10.10.0/24 -j ACCEPT
-A ufw-before-forward -d 10.10.10.0/24 -j ACCEPT
```

Save and close the file. Then enable UFW.

```
sudo ufw enable
```

If you have enabled UFW before, then you can use systemctl to restart UFW.

```
sudo systemctl restart ufw
```

Now if you list the rules in the POSTROUTING chain of the NAT table by using the following command:

```
sudo iptables -t nat -L POSTROUTING
```

You can see the Masquerade rule.

```
linuxbabe@buster: ~
linuxbabe@buster:~$ sudo iptables -t nat -L POSTROUTING
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
linuxbabe@buster:~$
```

Step 6: Install a DNS Resolver on the Server

Since we specified the VPN server as the DNS server for client, we need to run a DNS resolver on the VPN server. We can install the bind9 DNS server.

```
sudo apt install bind9
```

Once it’s installed, BIND will automatically start. You can check its status with:

```
systemctl status bind9
```

Sample output:

```
● named.service - BIND Domain Name Server
     Loaded: loaded (/lib/systemd/system/named.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2020-05-17 08:11:26 UTC; 37s ago
       Docs: man:named(8)
   Main PID: 13820 (named)
      Tasks: 5 (limit: 1074)
     Memory: 14.3M
     CGroup: /system.slice/named.service
             └─13820 /usr/sbin/named -f -u bind
```

If it’s not running, start it with:

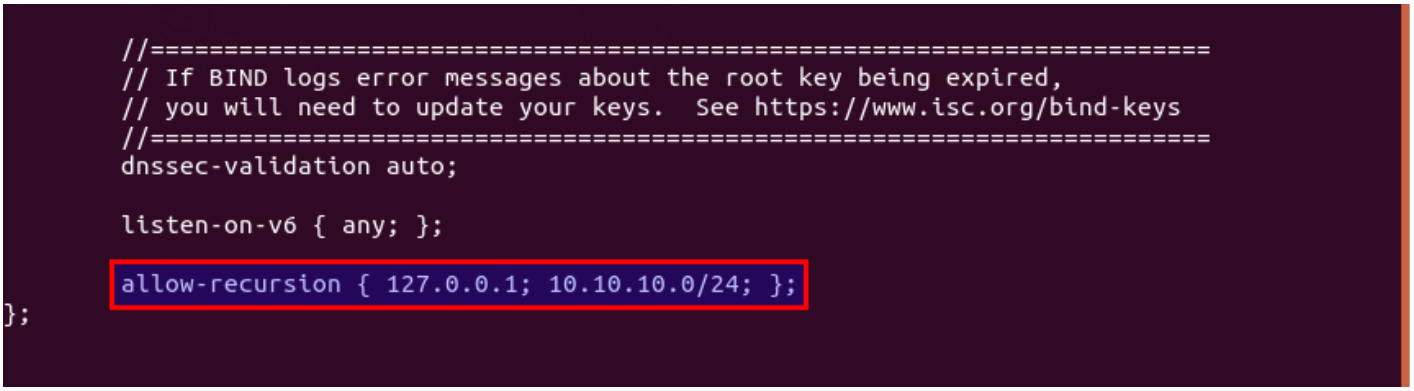
```
sudo systemctl start bind9
```

Edit the BIND DNS server’s configuration file.

```
sudo nano /etc/bind/named.conf.options
```

Add the following line to allow VPN clients to send recursive DNS queries.

```
allow-recursion { 127.0.0.1; 10.10.10.0/24; };
```



Save and close the file. Restart BIND9 for the changes to take effect.

```
sudo systemctl restart bind9
```

Then you need to run the following command to allow VPN clients to connect to port 53.

```
sudo ufw insert 1 allow in from 10.10.10.0/24
```

Step 7: Open WireGuard Port in Firewall

Run the following command to open UDP port 51820 on the server.

```
sudo ufw allow 51820/udp
```

Step 8: Start WireGuard server

Run the following command on the server to start WireGuard.

```
sudo wg-quick up /etc/wireguard/wg0.conf
```

To stop it, run

```
sudo wg-quick down /etc/wireguard/wg0.conf
```

You can also use systemd service to start WireGuard.

```
sudo systemctl start wg-quick@wg0.service
```

Enable auto-start at system boot time.

```
sudo systemctl enable wg-quick@wg0.service
```

Check its status with the following command. Its status should be `active (exited)`.

```
systemctl status wg-quick@wg0.service
```

Now WireGuard server is ready to accept client connections.

Client

Start WireGuard.

```
sudo systemctl start wg-quick@wg-client0.service
```

Enable auto-start at system boot time.

```
sudo systemctl enable wg-quick@wg-client0.service
```

Check its status:

```
systemctl status wg-quick@wg-client0.service
```

Now go to this website: `http://icanhazip.com/` to check your public IP address. If everything went well, it should display your VPN server's public IP address instead of your client computer's public IP address.

You can also run the following command to get the current public IP address.

```
curl https://icanhazip.com
```

Troubleshooting Tips

You can ping from the VPN server to VPN client (`ping 10.10.10.2`) to see if the tunnel works. If you see the following error message in the ping,

```
ping: sendmsg: Required key not available
```

it might be that the `AllowedIPs` parameter is wrong, like a typo.

If the VPN tunnel is successfully established, but the client public IP address doesn't change, that's because the masquerading or forwarding rule in your UFW config file is not working. I once had a typo in the `/etc/ufw/before.rules` file, which caused my computer not being able to browse the Internet.

Note that I don't recommend using `SaveConfig=true` in the `[Interface]` section of the WireGuard configuration file. `SaveConfig` tells WireGuard to save the runtime configuration on shutdown. So if you add additional `[Peer]` in the configuration file and then restart WireGuard, your newly-added configs will be overwritten.

If your VPN still doesn't work, try restarting the VPN server.

```
sudo systemctl restart wg-quick@wg0.service
```

Then stop the VPN client.

```
sudo systemctl stop wg-quick@wg-client0.service
```

And upgrade software packages on the VPN client.

```
sudo apt update; sudo apt upgrade
```

Next, reboot the VPN client.

```
sudo shutdown -r now


sudo systemctl start wg-quick@wg-client0.service
```

Adding Additional VPN Clients

WireGuard is designed to associate one IP address with one VPN client. To add more VPN clients, you need to create a unique private/public key pair for each client, then add each VPN client's public key in the server's config file (`/etc/wireguard/wg0.conf`) like this:

```

```

```
[Interface]
Address = 10.10.10.1/24
PrivateKey = UIFH+XXjJ0g0uAZJ6vPqsbb/o68SYVQdmYJpy/FlGFA=
ListenPort = 51820

[Peer]
PublicKey = 75VNV7HqFh+3QIT5OHZkcjWfbjx8tc6Ck62gZJT/KRA=
AllowedIPs = 10.10.10.2/32

[Peer]
PublicKey = YYh4/1Z/3rtl0i7cJorcinB7T4UOIzScifPNEIESFD8=
AllowedIPs = 10.10.10.3/32

[Peer]
PublicKey = EVstHZc6QamzPgefDGPLFEjGyedJk6SZbCJttpzcvC8=
AllowedIPs = 10.10.10.4/32
```

Each VPN client will have a static private IP address (10.10.10.2, 10.10.10.3, 10.10.10.4, etc). Restart the WireGuard server for the changes to take effect.

```
sudo systemctl restart wg-quick@wg0.service
```

Then add WireGuard configuration on each VPN client as usual.

Policy Routing, Split Tunneling & VPN Kill Switch

Now I will show you how to use **policy routing**, **split tunneling**, and **VPN kill switch** with WireGuard VPN. **Note** that it’s not recommended to use them in conjunction with each other. If you use policy routing, then you should not enable split tunneling or VPN kill switch, and vice versa.

Policy Routing

By default, all traffic on the VPN client will be routed through the VPN server. Sometimes you may want to route only a specific type of traffic, based on the transport layer protocol and the destination port. This is known as policy routing.

Policy routing is configured on the client computer, and we need to stop the WireGuard client

process.

```
sudo systemctl stop wg-quick@wg-client0.service
```

Then edit the client configuration file.

```
sudo nano /etc/wireguard/wg-client0.conf
```

For example, if you add the following 3 lines in the `[interface]` section, then WireGuard will create a routing table named “1234” and add the ip rule into the routing table. In this example, traffic will be routed through VPN server only when TCP is used as the transport layer protocol and the destination port is 25, i.e, when the client computer sends emails.

```
Table = 1234
PostUp = ip rule add ipproto tcp dport 25 table 1234
PreDown = ip rule delete ipproto tcp dport 25 table 1234
```

Save and close the file. Then start WireGuard client again.

```
sudo systemctl start wg-quick@wg-client0.service
```

Split Tunneling

By default, all traffic on the VPN client will be routed through the VPN server. Here’s how to enable split tunneling, so only traffic to the `10.10.10.0/24` IP range will be tunneled through WireGuard VPN. This is useful when you want to build a private network for several cloud servers, because VPN clients will run on cloud servers and if you use a full VPN tunnel, then you will probably lose connection to the cloud servers.

Edit the client configuration file.

```
sudo nano /etc/wireguard/wg-client0.conf
```

Change

```
AllowedIPs = 0.0.0.0/0
```

To

```
AllowedIPs = 10.10.10.0/24
```

So traffic will be routed through VPN only when the destination address is in the 10.10.10.0/24 IP range. Save and close the file. Then restart WireGuard client.

```
sudo systemctl restart wg-quick@wg-client0.service
```

VPN Kill Switch

By default, your computer can access the Internet via the normal gateway when the VPN connection is disrupted. You may want to enable the kill switch feature, which prevents the flow of unencrypted packets through non-WireGuard interfaces.

Stop the WireGuard client process.

```
sudo systemctl stop wg-quick@wg-client0.service
```

Edit the client configuration file.

```
sudo nano /etc/wireguard/wg-client0.conf
```

Add the following two lines in the `[interface]` section.

```
PostUp = iptables -I OUTPUT ! -o %i -m mark ! --mark $(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT
PreDown = iptables -D OUTPUT ! -o %i -m mark ! --mark $(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT
```

Like this:

```
[Interface]
```

```
Address = 10.10.10.2/24
DNS = 10.10.10.1
PrivateKey = cOFA+x5UvHF+a3xJ6enLatG+DoE3I5PhMgKrMKkUyXI=
PostUp = iptables -I OUTPUT ! -o %i -m mark ! --mark $(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT
PreDown = iptables -D OUTPUT ! -o %i -m mark ! --mark $(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT

[Peer]
PublicKey = kQvxOJI5Km4S1c7WXu2UZFpB8mHGuf3Gz8mmgTIF2U0=
AllowedIPs = 0.0.0.0/0
Endpoint = 12.34.56.78:51820
PersistentKeepalive = 25
```

Save and close the file. Then start the WireGuard client.

```
sudo systemctl start wg-quick@wg-client0.service
```

Installing Linux Kernel 5.x on Debian 10

The current Linux kernel version on Debian 10 is 4.19. In step 1, we added the backport repository on Debian 10. The **backport repository** includes Linux kernel 5.8, at the time of this writting. You may probably know that the wireguard module is included in the Linux kernel starting from version 5.4. If we install Linux kernel 5.8 on Debian 10, we don't need to build the wireguard module when the system is upgrading the Linux kernel. As a matter of fact, my Debian 10 server once had a problem in building the wireguard module with wireguard-dkms.

Note that by the time you read this article, the Debian 10 backport repository might have removed kernel 5.8 and included kernel 5.9. Simply repalce 5.8 with 5.9 in the following commands.

To install Linux kernel 5.8 on Debian 10 cloud servers, run the following command.

```
sudo apt install linux-image-5.8.0-0.bpo.2-cloud-amd64 linux-headers-5.8.0-0.bpo.2-cloud-amd64
```

To install Linux kernel 5.8 on a Debian 10 PC, run the following command.

```
sudo apt install linux-image-5.8.0-0.bpo.2-amd64 linux-headers-5.8.0-0.bpo.2-amd64
```

Then restart your Debian 10 box.

```
sudo shutdown -r now
```

Check your Linux kernel version.

```
uname -r
```

Sample output

```
5.8.0-0.bpo.2-cloud-amd64
```

Although we no longer need the `wireguard-dkms` package, it's a dependency for the `wireguard` package, so we can't remove it from the system. You will probabaly see the following error when upgrading the `wireguard` package.

```
Error! The dkms.conf for this module includes a BUILD_EXCLUSIVE directive which
does not match this kernel/arch. This indicates that it should not be built
```

This indicates `wireguard-dkms` is trying to build the `wireguard` module into the Linux kernel, but Linux 5.8 includes a native `wireguard` module, so the build operation is prevented and you can ignore this error.

Revision #1
Created 11 July 2021 19:13:57 by Admin
Updated 11 July 2021 19:14:25 by Admin