Part 1 – Installation and server configuration

Introduction

You have one or more services running on a VPS somewhere. Because the VPS is sitting on the internet with a public IP, the easiest way to access it is by exposing a port on that VPS where the service is running. This, however, opens up a load of possible security issues. Because now anyone on the internet can access the service just like you. This is perfectly fine in most cases, I mean, all websites that you browse work this way – they are exposed so that people can access them. You may have some form of 2FA or other application-level security method setup, but that doesn't help mitigate vulnerabilities in the application itself. By limiting access to your application in the first place, you covering a large part of the attack surface. **All of this by leveraging the power of VPN technology.**

Setup Plan

Currently, all services on my VPS are listening on the server's public IP, which is the only network interface together with localhost. Here's what we will do:

1. Install and setup Wireguard interface on the server
2. Setup clients
3. Stop all services and bind them only to the Wireguard interface
4. Adjust firewall rules according to our new setup

> This guide assumes that you will access these services only from a few devices. If, for example, you would like to provide your whole home network with access to the services running on your VPS, you will have to do it a bit differently.

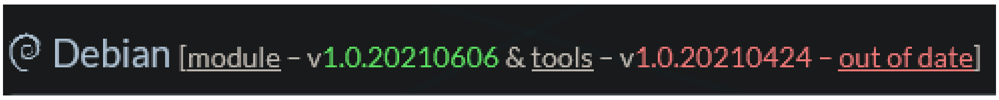
```
|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|□□□□|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|
|□□□□□□HOME□□□□□□|□□□□|□□□□□VPS□□□□□□|
|Android Device------------------------------------>---WG tunnel--->□□□□□□□□□□□|
|[Interface]□□□□□□□□□□|□□□□|□□□□□□□□□□□|
|eth0: 192.168.20.55 (private IP from DHCP at home)□|□□□□|Server  □□□□□□□□□|
|WG IP: 10.20.20.2/29□□□□□□□□|□□□□|[Interface]□□□□□□□□|
|WG public key: 16f5das48wa1f684g1a489awg5a□□□|□□□□|eth0: 78.97.52.14 (example public IP)□□|
|WG private key: gzf74894ger89a46sd14g84r8esg□□|□□□□|WG IP: 10.20.20.1/29□□□□□□|
|□□□□□□□□□□□□□|□□□□|WG public key: 6t57489hgnufjfiosdjjfp98h□|
|Windows Device------------------------------------>---WG tunnel--->WG private key: 4gfd89a7g1fd56g848g4fdg41fd|
|[Interface]□□□□□□□□□□|□□□□|□□□□□□□□□□□|
|eth0: 192.168.20.56 (private IP from DHCP at home)□|□□□□|□□□□□□□□□□□|
|WG IP: 10.20.20.3/29□□□□□□□□|□□□□|□□□□□□□□□□□|
|WG public key: fgd489fdsg84168e46g1514ge5g□□□|□□□□|□□□□□□□□□□□|
|WG private key: iyut789tr496516sh416g4164h6h□□|□□□□|□□□□□□□□□□□|
|________________________________________________|□□□□|________________________________________|
```

*note: don't worry, the public and private keys in the diagram are just random placeholder values

Installation on the VPS

Since we are running Debian, we can get Wireguard from the official repository. I usually prefer building software from source, but for the sake of this guide, I will go the easier route and simply use the repo version.

> Note that since it's Debian, the packages are sometimes a bit outdated. At the time of writing, the tools weren't available in the newest version – even in the unstable repo. The default stable repo has even older packages.

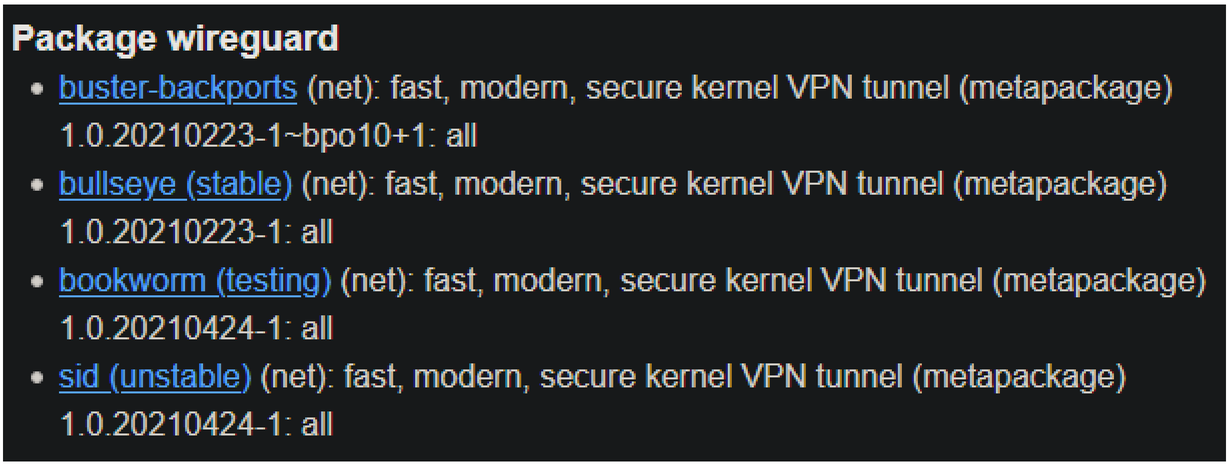


Debian [module – v1.0.20210606 & tools – v1.0.20210424 – out of date]

The situation around Debian and Wireguard is a bit confusing. Debian is known to be stable but has older packages. If you don't want to worry about anything, just install it from the stable repo:

```
$ sudo apt install wireguard
```

However, if you are on Debian 10, Wireguard still isn't integrated into the 4.19 kernel, which means the installation will bring the wireguard-dkms (Dynamic Kernel Module Support) package as well. On Debian 11, this should not be necessary, because Wireguard is already natively in the 5.10 kernel (which is default for Debian 11). I have it a bit more complicated. I am in fact running Debian 11, but with the 5.4 kernel (backported from Debian 10), which also doesn't have Wireguard natively, but that's just a side note. You can see the version status of the Wireguard package here:



Package wireguard

- buster-backports (net): fast, modern, secure kernel VPN tunnel (metapackage)
  1.0.20210223-1~bpo10+1: all
- bullseye (stable) (net): fast, modern, secure kernel VPN tunnel (metapackage)
  1.0.20210223-1: all
- bookworm (testing) (net): fast, modern, secure kernel VPN tunnel (metapackage)
  1.0.20210424-1: all
- sid (unstable) (net): fast, modern, secure kernel VPN tunnel (metapackage)
  1.0.20210424-1: all

To find out how to add testing or unstable to Debian 11, check out my guide over here.

Configure Wireguard on the server

The main configuration folder is located in `/etc/wireguard`. This directory will contain both configuration and private/public key, therefore it is only accessible with root by default. To get to this directory, you need to elevate privileges.

```
$ sudo su
(root)$ cd /etc/wireguard/
```

Create configuration file for the Wireguard interface

Create a new file in `/etc/wireguard`. The name of the file will also be the name of the interface (like eth0,lo,etc.) + .conf file extension. I like the default naming, so I will use name it `wg0.conf`. Before creating the file, change `umask` to `077`, so that the file is readable only by root. `Umask` controlls what permissions will newly created files and directories have. The default umask is `022`. When we use the `umask` command with a different number, all new files and directories will be created under permissions we have set. This modifies the default `umask` value only for the current sub-shell. To go back to the default `umask`, just logout/login. You can also make the changes permanent if you want. You can read more about `umask` here.

```
(root)$ umask 077


(root)/etc/wireguard$ touch wg0.conf
```

Prepare wg0.conf

Open `wg0.conf` with your favorite editor.

```
(root)$ vi /etc/wireguard/wg0.conf
```

We can slowly start preparing the config file, step by step. Right now, add the `[Interface]` block, which will define properties of the server's Wireguard interface.

- `Address` – The local IP of the server within the Wireguard tunnel and subnet mask in CIDR notation. I know I will only have 3 clients, therefore I picked a small subnet `/29` with only 6 usable hosts. (Basic subnetting knowledge is required).
- `ListenPort` – The port on the server Wireguard will use to communicate with other peers. After everything is set up, this will be the only port exposed to the open internet. If left unspecified, default is `51820`.
- `PrivateKey` – The private key of the server that will be generated in a moment. **This key should never be shared or leave the server. We will only send out the public key** (as the name suggests).

> **Always keep private keys on devices where they were created. They should never be moved accross network or even accross devices.**

The `wg0.conf` file should look like this right now:

```
[Interface]
Address = 10.20.20.1/29
ListenPort = 51895
PrivateKey =
```

Generate server public/private keys

Wireguard is built around the public and private key pairs structure. There is technically no such a thing as *client* or *server* in Wireguard, because everything is a *peer*. We use the *client/server* terminology to help our mind imagine the setup better. Each *peer* will generate its own private and public key. The **private key** will never leave the device where it originated from and should be kept well secured. The **public key** will be copied over to the other peers. In my case, the VPS will become a *peer* to both the Android and Windows device (*clients*). However, Android nor Windows will even know about each other, they will only be aware of the VPS, which will be their only *peer*. Reason for this is that they only need to talk to the VPS, not to each other.

Still as root, with `umask` set to `0077` (check by typing `umask` in the terminal) and in the `/etc/wireguard` directory, create a new subdirectory for the public/private keys and preshared keys (will explain in a second).

```
(root)$ mkdir keys psk
```

Navigate to the `keys` directory and generate the server key pair.

```
(root)$ cd keys
(root)$ wg genkey | tee wg0_private.key | wg pubkey > wg0_public.key
```

The `wg genkey` command generates a random *private* key in base64 and prints it to standard output (terminal). The output is instead redirected to `tee`, which both prints it to stdout (terminal), but also saves it into a file `wg0_private.key`. The private key printed to stdout is then piped (`|` symbol) to `wg pubkey`, which calculates the public key and prints it in base64 to stdout from a corresponding private key (the one we redirected to it with the pipe), lastly redirect the public key from stdout to a file `wg0_public.key`

Now we have two files in the `keys` directory:

```
(root)$ ls


wg0_private.key  wg0_public.key
```

Generate preshared keys

> This option adds an additional layer of symmetric-key cryptography to be mixed into the already existing public-key cryptography, for post-quantum resistance (for paranoid people like me).

From the Wireguard whitepaper:

> “ In order to mitigate any future advances in quantum computing, WireGuard also supports a mode in which any pair of peers might additionally pre-share a single 256-bit symmetric encryption key between themselves, in order to add an additional layer of symmetric encryption. The attack model here is that adversaries may be recording encrypted traffic on a long term basis, in hopes of someday being able to break Curve25519 and decrypt past traffic. While pre-sharing symmetric encryption keys is usually troublesome from a key management perspective and might be more likely stolen, the idea is that by the time quantum computing advances to break Curve25519, this pre-shared symmetric key has been long forgotten. And, more importantly, in the shorter term, if the pre-shared symmetric key is compromised, the Curve25519 keys still provide more than sufficient protection. In lieu of using a completely post-quantum crypto system, which as of writing are not practical for use here, this optional hybrid approach of a pre-shared symmetric key to complement the elliptic curve cryptography provides a sound and acceptable trade-off for the extremely paranoid. Furthermore, it allows for building on top of WireGuard sophisticated key-rotation schemes, in order to achieve varying types of post-

> compromise security.

Move to the `psk` directory and generate preshared keys for each of the clients. The preshared key will then be part of both the server and client configuration file.

```
(root)$ cd /etc/wireguard/psk
(root)$ wg genpsk > android_client.psk
(root)$ wg genpsk > windows10_client.psk
```

Again, there should be 2 files in the directory, one for each client.

```
(root)$ ls

android_client.psk  windows10_client.psk
```

Edit wg0.conf

Open the config file again and fill in some of the remaining information:

```
[Interface]
Address = 10.20.20.1/29
ListenPort = 51895
PrivateKey = your_private_key

[Peer]
PublicKey =
PresharedKey = your_preshared_key_for_this_client
AllowedIPs = 10.20.20.2/32

[Peer]
PublicKey =
PresharedKey = your_preshared_key_for_this_client
AllowedIPs = 10.20.20.3/32
```

> Open two terminals – one with `wg0.conf` in editor and the other in `/etc/wireguard/` directory. Now use `cat` to print contents of your newly generated key files to the terminal and simply copy and paste them to the editor.

`AllowedIPs` – IP address within the Wireguard tunnel for each peer in CIDR notation. Wireguard will accept only packets coming from these specific IP ranges or from a single IP of the client when used with `/32`.

This is it for now, we will come back to the config file in a moment.

> Remember which peer is which. The first one is for the Android device, while the second one is for the Windows client.

Revision #3
Created 22 September 2021 01:51:32 by Marek
Updated 22 September 2021 01:57:28 by Marek