SSH over Wireguard VPN

Managing servers on the public internet brings a lot of security threats with it. RDP is so bad it isn't recommended to even expose it to the public internet. With SSH, things are a bit more safe, but only after you perform a set of SSH hardening tasks, e.g.:

- Change default SSH port (security through obscurity, only helps reduce number of bots attempting to connect)
- Disable root login
- Disable password login, only use PubKey authentication
- Setup 2FA for SSH
- etc.

In order to add another layer of protection, you can setup VPN to protect your SSH connections (while leaving your website available to the public internet).

> Having SSH over VPN shouldn't make you forget about the aforementioned SSH hardening steps. VPN is just one of them.

Wireguard Server setup

> This guide uses Debian 11 as the server and Windows 10 as the client.

Install Wireguard on the server

Debian offers fairly outdated version of Wireguard in the official `stable` repository. At the time of writing, `stable` has `20210223`, while `testing` has `20210424`. You can check available version with `apt`.

```
$ apt-cache policy wireguard
```

Output:

```
wireguard:
  Installed: (none)
  Candidate: 1.0.20210223-1
  Version table:
     1.0.20210223-1 500
```

```
    500 https://ftp.sh.cvut.cz/debian bullseye/main amd64 Packages
```

Install from stable

You can decide if you want to use the `stable` or `testing` version. To install `stable`, simply type:

```
$ sudo apt install wireguard
```

Install from testing

To install from the `testing` repo, you have to perform the steps shown HERE. **TLDR**, add testing to `/etc/apt/sources.list`, adjust `apt` preferences in `/etc/apt/preferences` and update `apt`. Once you have enabled the testing repository with appropriate priorities, you can install Wireguard. To see if the preferences are set properly, try to see what apt would install. By default, everything should be installed from `stable`, unless specified otherwise.

```
$ apt-cache policy wireguard
```

`Candidate` line shows what would be installed.

```
wireguard:
  Installed: (none)
  Candidate: 1.0.20210223-1
  Version table:
    1.0.20210424-1 -10
      -10 http://deb.debian.org/debian testing/main amd64 Packages
    1.0.20210223-1 900
      900 https://ftp.sh.cvut.cz/debian stable/main amd64 Packages
```

Now specify that you want to install from `testing`:

```
$ apt-cache policy -t testing wireguard
```

`Candidate` should now point to the newer version:

```
wireguard:
  Installed: (none)
  Candidate: 1.0.20210424-1
  Version table:
    1.0.20210424-1 990
      990 http://deb.debian.org/debian testing/main amd64 Packages
    1.0.20210223-1 900
```

```
    900 https://ftp.sh.cvut.cz/debian stable/main amd64 Packages
```

Finally, to install from `testing` (`-t` to specify repository to install from), type this:

```
$ sudo apt install -t testing wireguard
```

Generate keys

Regardless of the version you have installed, it's time to create the configuration, directories and keys. We will be working in a restricted directory (only readable by `root`), so elevate the shell:

```
$ sudo su
(root)$
```

Create directories

We need some place to store our public, private and preshared keys. All of these should be located in `/etc/wireguard` directory and only readable by `root`. The directory structure itself is up to you, I prefer having directory `keys` in `/etc/wireguard` for server public and private key and `psk` directory for preshared keys.

Set `umask` to `077` to create files and directories only readable by root:

```
(root)$ umask 077
```

Create the aforementioned directories in `/etc/wireguard`:

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

Generate server keys

Navigate to the `/etc/wireguard/keys` directory and generate the keys.

```
(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`

You will now have two files in `/etc/wireguard` directory. One containing public, the other private key.

```
wg0_private.key  wg0_public.key
```

Create server configuration

Still under `umask 077` and in the `root` shell, create a config file in `/etc/wireguard` directory. The name of the file will be used as the name of the interface.

```
(root)$ touch wg0.conf
```

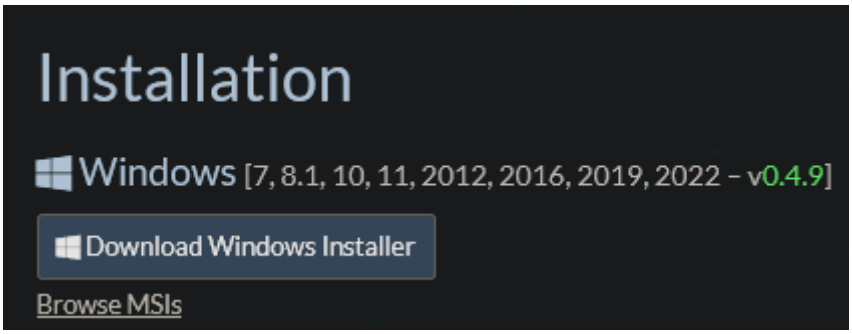
Replace `server_private_key` with the private key of your server (content of `wg0_private.key` in `/etc/wireguard/keys`). `ListenPort` is the port you want Wireguard to listen on and `Address` specifies the subnet for the Wireguard tunnel. I chose `/29` due to small subnet size, because more hosts aren't necessary.

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

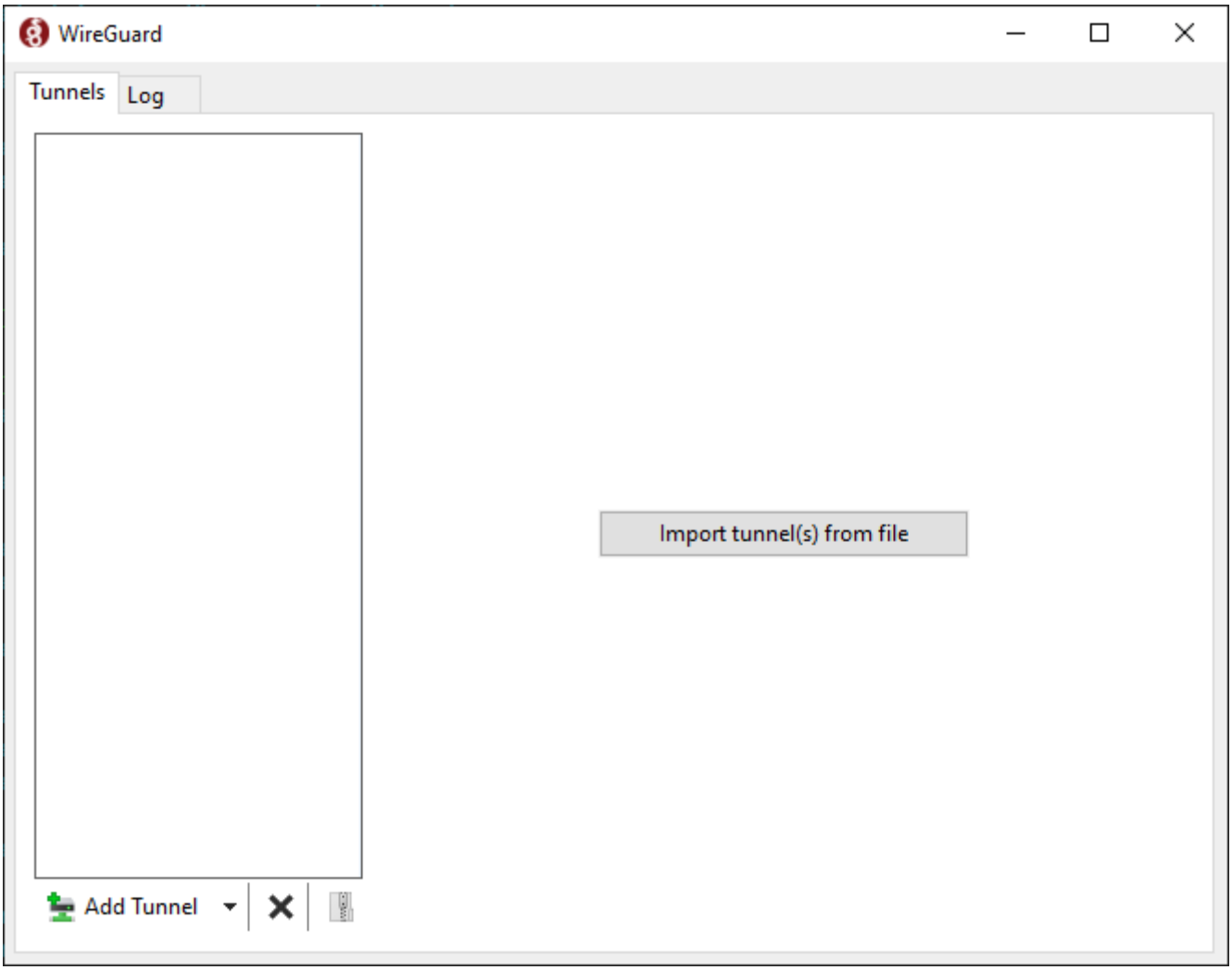
Client Wireguard setup

Download and install Wireguard

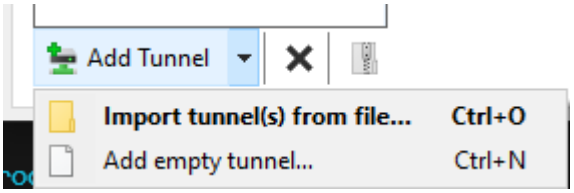
Download the Windows Installer from the official Wireguard website.



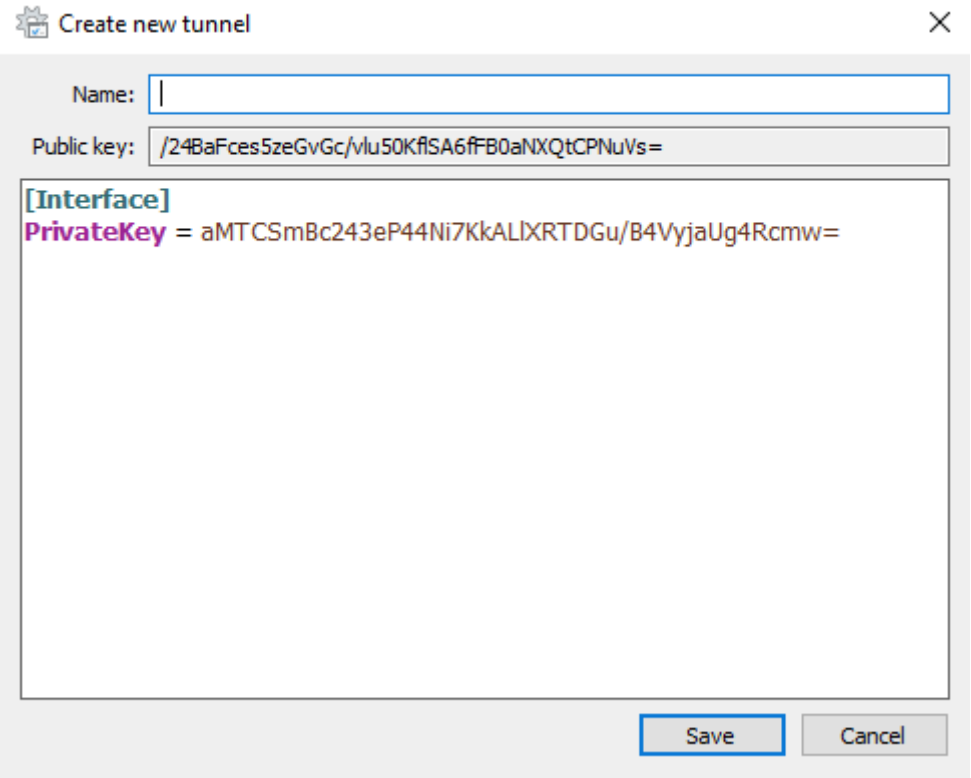
Run the `wireguard-installer.exe` and after a few moments, this window should appear:



Click the arrow next to `Add Tunnel` and select `Add empty tunnel...`



Wireguard will automatically generate a private and public key for this client.



*note: all keys shown will be destroyed afterwards and are only used for demonstration purposes

Configure the client

Under the `PrivateKey` line, add `Address`, which will be the address of the Wireguard client within the Wireguard tunnel.

```
[Interface]
PrivateKey = private_key
Address = 10.20.20.2/29
```

Even though I am refering to the *server* as *Wireguard server* and my management station as the *client*, the truth is Wireguard doesn't care about that. In it's eyes, **both are on the same level and there's no difference between them, they are both *peers*.**

To configure the *server*, add `[Peer]` block to the *client* config.

```
[Peer]
PublicKey = server_public_key
AllowedIPs = 10.20.20.1/32
Endpoint = publicIP:port
```

- `PublicKey` – Copy the content of `/etc/wireguard/wg0_public.key` to this line
- `AllowedIPs` – IP of the server within the tunnel, use `/32` to allow only the server and not the whole subnet

- `Endpoint` – Public IP of the server and port where Wireguard is running (same as `ListenPort` in the server `wg0.conf`)

The entire client config should now look like this:

```
[Interface]
PrivateKey = private_key
Address = 10.20.20.2/29

[Peer]
PublicKey = server_public_key
AllowedIPs = 10.20.20.1/32
Endpoint = publicIP:port
```

Add client to the server configuration

To finish setting up the tunnel, add the client's public key and IP address to the `wg0.conf` in `/etc/wireguard`.

```
[Peer]
PublicKey = public_key_from_client
AllowedIPs = 10.20.20.2/32
```

You have to copy the public key from the client to the server config. **Make sure you don't switch up the public and private keys. Always copy the public key only! The private key should never leave the device it was created on.**

`wg0.conf` will look something like this right now:

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

[Peer]
PublicKey = public_key_from_client
AllowedIPs = 10.20.20.2/32
```

Restart interface to refresh config

After editing `wg0.conf`, always run `wg-quick` to restart the interface, otherwise the changes to the configuration won't propagate.

```
$ sudo wg-quick down wg0
[#] ip link delete dev wg0


$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.20.20.1/29 dev wg0
[#] ip link set mtu 1420 up dev wg0
```

> The tunnel should be now set up. Try pinging the server from the client. If it isn't working, make sure firewall like `iptables` isn't blocking the communication and you have ICMP enabled.

(Optional) Secure with Preshared keys

To add additional level of security, you can generate prehared keys for the client, which act as a shared secret and provide another layers of cryptographical security.

Generate Preshared keys

Navigate to the `/etc/wireguard/psk` directory, set `umask` to `077` and generate the key (still in the `root` shell):

```
(root)$ cd /etc/wireguard/psk
(root)$ umask 077
(root)$ wg genpsk > windows10_client.psk
```

Add to server config

Open `wg0.conf` and add `PresharedKey` to the `[Peer]` block for the client:

```
[Peer]
PublicKey = public_key_from_client
PresharedKey = preshared_key
AllowedIPs = 10.20.20.2/32
```

Add to client config

Because it's a shared secret, it has to be added both to the server and to the client. Also add `Presharedkey` to the `[Peer]` block.

```
[Interface]
PrivateKey = private_key
Address = 10.20.20.2/29

[Peer]
PublicKey = server_public_key
PresharedKey = preshared_key
AllowedIPs = 10.20.20.1/32
Endpoint = publicIP:port
```

Move SSH to Wireguard interface

Test connection over Wireguard

Right now, SSH is listening on `0.0.0.0` which means all available interfaces. This means it should be listening on the Wireguard interface as well. Try connecting over the Wireguard IP.

```
> ssh user@10.20.20.1 -p SSHport
```

```
The authenticity of host '[10.20.20.1]:port ([10.20.20.1]:port)' can't be established.
ECDSA key fingerprint is SHA256:fJOIGJoidhifdshug543fhsdof8h43hfo4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.20.20.1]:port' (ECDSA) to the list of known hosts.
```

Bind SSH to Wireguard interface

Open `/etc/ssh/sshd_config` and set `ListenAddress` to the IP address of the Wireguard interface (`wg0`).

```
$ ip a
...
 wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
    link/none
    inet 10.20.20.1/29 scope global wg0
       valid_lft forever preferred_lft forever
```

```
$ sudo vi /etc/ssh/sshd_config
```

`/etc/ssh/sshd_config`

```
ListenAddress 10.20.20.1
```

Restart SSH and check:

```
$ sudo systemctl restart sshd


$ systemtl status sshd
...
Starting OpenBSD Secure Shell server...
Server listening on 10.20.20.1 port {ssh_port}.
Started OpenBSD Secure Shell server.
```

(Optional) Move Nginx to public IP only

The website (Nginx) is still listening on all interfaces on ports 80 and 443, but we only need it on the public IP, therefore:

- SSH on Wireguard IP ONLY
- Nginx on public IP ONLY

Open `/etc/nginx/conf.d/name_of_your_config.conf` and add the public IP of your server in front of port on the `listen` line. Both for 80 and 443.

```
server {
  listen 443 ssl;
```

to

```
server {
  listen publicIP:443 ssl;
```

Restart Nginx:

```
$ sudo systemctl restart nginx
```

Check with netstat that Nginx is listening on the correct IP:

```
$ sudo netstat -tulpn
...
tcp        0      0 publicIP:80          0.0.0.0:*               LISTEN      27870/nginx: master
tcp        0      0 publicIP:443         0.0.0.0:*               LISTEN      27870/nginx: master
```

Firewall setup

Wireguard tunnel is now fully established, SSH listens on the Wireguard interface, Nginx on the public IP. Let's adjust the firewall (`iptables`). I use `iptables-restore` in combination with a configuration file in `/etc/iptables/rules.v4`. The file should look like this, explaining what each of the lines is for is out of the scope of this guide, brief comment can be found on each line:

```
*filter

# Drop forwarded traffic, we don't need that since we are not acting as a router
:FORWARD DROP [0:0]

# Accept all outgoing connection now, later it will be restricted
:OUTPUT ACCEPT [0:0]

# Block all incoming traffic
:INPUT DROP [0:0]

# Do not block localhost
-A INPUT -i lo -j ACCEPT

# Allow established and related incoming connections
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Allow SSH on the port that it's running on
-A INPUT -i wg0 -p tcp -m tcp --dport SSH_PORT -j ACCEPT

# Allow web traffic
-A INPUT -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 443 -j ACCEPT

# Allow Wireguard on public IP
-A INPUT -p udp -m udp --dport WIREGUARD_PORT -j ACCEPT

# Allow ping on all interfaces
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT

# Commit rules
COMMIT
```

Commit rules with `iptables-restore`:

```
$ sudo iptables-restore /etc/iptables/rules.v4
```

> That's it, you now have SSH secured inside of a VPN tunnel and the website is still accessible over the public internet.

Enable Wireguard on startup

Because we have limited SSH only to our VPN, we need it running all the time to access the server. In the current state, if the server restarts, Wireguard won't start up and we won't be able to connect.

Create wg0.service

I am running Debian where services are managed by systemd. Fortunately, we don't have to create a Unit file from scratch, just issue one command:

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

```
Created symlink /etc/systemd/system/multi-user.target.wants/wg-quick@wg0.service → /lib/systemd/system/wg-quick@.service.
```

Reload daemon service to apply changes:

```
$ sudo systemctl daemon-reload
```

Make sure Wireguard starts before SSH

If you had tried to reboot right now, you would find out that you cannot connect. That is because SSHd would most likely try to start before Wireguard interface is ready. This will fail, because it would try binding to an interface that doesn't exist yet. To fix this, we would need to edit the SSHd service Unit file:

```
$ sudo systemctl edit --full sshd
```

On the `After` line, add `wg-quick@wg0.service` to make sure SSHd starts after `wg-quick@wg0.service`.

```
After=network.target auditd.service wg-quick@wg0.service
```

You can also make another adjustment by adding the line `Requires` and specifying the `wg0` virtual interface like this:

```
Requires=sys-devices-virtual-net-wg0.device
```

Restart SSHd

```
$ sudo systemctl restart sshd
```

> This will make sure SSHd starts after the Wireguard interface is ready.

> If you have `iptables` as your firewall, make sure the rules are applied after startup, research `iptables-persistent` to learn more.

Revision #12
Created 29 September 2021 01:02:55 by Marek
Updated 29 September 2021 13:14:36 by Marek