Part 3 – Migrating services on VPS

Moving network services to the WG interface

In order to move our network services to another interface, we first need to find out what exactly is running on the server. We are interested in services and their listening ports. There are a few commands to achieve this, e.g `netstat`, `lsof`, `ss` and `nmap`. Pick the one that works on your server, so you don't have to install anything additional just for one command.

```
$ netstat -tulpn
```

I will run netstat with a couple of flags grouped together, definitely easier than writing `netstat -t -u -l -p -n`

- `-t` lists TCP
- `-u` lists UDP
- `-l` shows only listening ports (omitted by default)
- `-p` shows what program/service is running on each port
- `-n` doesn't resolve IPs and hosts

```
$ sudo netstat -tulpn

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      72/systemd-resolved
tcp        0      0 127.0.0.1:8888          0.0.0.0:*               LISTEN      205/uwsgi
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      98/nginx: master pr
tcp        0      0 127.0.0.1:4004          0.0.0.0:*               LISTEN      82/filtron
tcp        0      0 127.0.0.1:4005          0.0.0.0:*               LISTEN      82/filtron
tcp        0      0 127.0.0.1:5000          0.0.0.0:*               LISTEN      77/python3
tcp        0      0 78.97.52.14:7985        0.0.0.0:*               LISTEN      107/sshd: /usr/sbin
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      98/nginx: master pr
udp        0      0 127.0.0.53:53           0.0.0.0:*                           72/systemd-resolved
udp        0      0 0.0.0.0:51895           0.0.0.0:*                           -
udp6       0      0 :::51895                :::*                                -
```

We can see a few services running on localhost (`127.0.0.1`), systemd-resolve localhost address (`127.0.0.53`), example public IP (`78.97.52.14`) or on all interfaces (`0.0.0.0`). The only one that we will care about now are `0.0.0.0` and `78.97.52.14`. From now on, everything will be very setup specific, so I might even change the way I write, from *we* to *me.*

Binding Nginx to wg0 interface

The main service I want to move is Nginx. It runs all my websites on this server, so it would make sense to migrate it first. Right now, it binds itself to all interfaces (`0.0.0.0`), so the websites should technically be already accessible on the Wireguard IP of the server (`10.20.20.1`), but I want them to be ONLY accessible on that interface.

Edit `/etc/nginx/conf.d/name_of_config.conf` and add the Wireguard IP address in front of ports in `server` config block on the `listen` line, like this:

```
server {
    listen 443 ssl http2;
    server_name youwebsite.com
    ...
    ...

to

server {
    listen 10.20.20.1:443 ssl http2;
    server_name youwebsite.com
    ...
    ...
```

Before reloading or restarting Nginx, make sure Wireguard is running and the interface is up, otherwise you will run into errors.

```
$ sudo wg-quick up wg0
```

Reload Nginx to cause least interruption to the service.

```
$ sudo systemctl reload nginx
```

Run netstat again to check if Nginx rebinded like I wanted:

```
$ sudo netstat -tulpn
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      72/systemd-resolved
tcp        0      0 127.0.0.1:8888          0.0.0.0:*               LISTEN      205/uwsgi
tcp        0      0 10.20.20.1:443          0.0.0.0:*               LISTEN      23281/nginx: master
tcp        0      0 127.0.0.1:4004          0.0.0.0:*               LISTEN      82/filtron
tcp        0      0 127.0.0.1:4005          0.0.0.0:*               LISTEN      82/filtron
tcp        0      0 127.0.0.1:5000          0.0.0.0:*               LISTEN      77/python3
tcp        0      0 78.97.52.14:7985        0.0.0.0:*               LISTEN      107/sshd: /usr/sbin
tcp        0      0 10.20.20.1:80           0.0.0.0:*               LISTEN      23281/nginx: master
udp        0      0 127.0.0.53:53           0.0.0.0:*                           72/systemd-resolved
udp        0      0 0.0.0.0:51895           0.0.0.0:*                           -
udp6       0      0 :::51895                :::*                                -
```

When I try to access my websites now, it....doesn't work. But at least I perfectly know why and how to fix it. Before, when I wanted to access `mysite.com`, I typed in `mysite.com` into the browser address bar and my device requested a DNS record for `mysite.com`, which turned out to be a public IP address of my server, or something else (depending on what I had set on my domain's registrar website.) The DNS records still points to the public IP of my server, but Nginx isn't serving the websites on that public IP anymore, but on the Wireguard one. How do I fix this?

Remove old DNS records

First of all, I will remove the old DNS A records with my registrar so that I don't forget. This varies vastly from registrar to registrar, so you will have to figure it our on your own.

Point new DNS records to Wireguard

The problem is that I cannot just create new A records with the Wireguard IP (`10.20.20.1`), because that's a reserved private IP, it wouldn't route anywhere. There are couple of ways to solve this:

1. Fill in the `DNS Server` field on both clients with a DNS server that will be able to provide resolution for the private IP `10.20.20.1`. It could even be the `10.20.20.1` server itself, but that means I would have to setup a DNS service on the server
2. Point clients (using the `DNS Server` field) to your local DNS server on your home network and create the A records there. This method, however, will not work for roaming clients if they connect to a different network (like mobile data).
3. Since it's just one server with two websites and two clients (this method wouldn't make sense in larger deployments), I could just edit local DNS entries (hosts file) to avoid having to configure a whole new DNS server, just for this single purpose.

I might look into a more sophisticated solution for DNS with split tunnel VPN later on, but for now I will use the third method.

Editing hosts file on Windows

Navigate to `C:\Windows\System32\drivers\etc` in the Windows Explorer and open the file named `hosts` in Notepad and add the following lines into the file:

```
10.20.20.1□□mysite.com
10.20.20.1□□mysite2.com
```

You might need to flush your DNS cache, but after saving the file, those two sites should now resolve correctly, because Windows checks the local hosts file first by default.

Editing hosts file on Android

With Android, it's a little more complicated. Since its a mobile operating system designed to keep the user away from all advanced configuration (but still better than iPhone in this regard), we cannot simply open the hosts file and edit it. On a rooted phone you can, but on a regular one you can't. We have basically two options

1. Use a pseudo-VPN app to re-route all traffic locally through a filter that allows you to add custom DNS entries (I think Blokada can do that, apart from other things), except we can't, because `There can be only one VPN connection running at the same time. The existing interface is deactivated when a new one is created.` – developer.android.com Judging by one comment on Reddit "*I'm not entirely sure if this is how it works but using wireguard on a rooted phone and a compatible kernel let's wireguard interact directly through the kernel. It doesn't show up as an active VPN through android. So it may be possible to use that and a conventional VPN approach to have two at once?*" If this actually works, it should theoretically be possible, but you need a rooted phone and it's probably in early stages.
2. Use the VPN server as a DNS server, since it's a roaming device anyway.
3. ~~Use `adb` to pull the hosts file to PC, edit it and push it back to your Android device. That's what I'll do.~~

~~How to get `abd` up and running is outside of this guide, there might be a guide for that on this site at some point, but until then, refer to the internet.~~

~~Enable USB debugging in Android settings and connect it using a cable to you PC. Allow USB debugging for this specific device and check that it's recongnized.~~

```
$ .\adb.exe devices
List of devices attached
215e29da        device
```

~~Transfer the hosts file from Android to your device:~~

```
.\adb.exe pull /system/etc/hosts C:\Users\Marek\Desktop
```

~~Open it with Notepad and add new lines containing IP and hostname in this format:~~

```
10.20.20.1□□mysite.com
10.20.20.1□□mysite2.com
```

> Nope, this didn't work because pushing the file to a read-only file system doesn't work. Christ, I have to root my phone as soon as I can...

Revision #3
Created 22 September 2021 01:52:18 by Marek
Updated 22 September 2021 01:58:20 by Marek