Playing with WireGuard
HW7 - CNS Sapienza

Gianfranco Romani 1814407

21 December 2020

1 Introduction

In this report I will describe how to set WireGuard to create a VPN for two machines and problems and results I obtained from some tests.

2 WireGuard

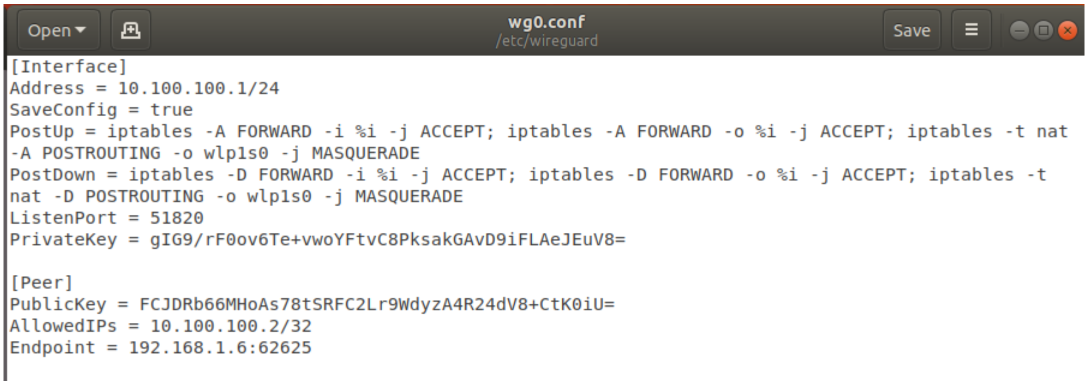
WireGuard is an open-source software used to configure and implement Virtual Private Networks. It is considered to be faster, simpler and more powerful than other VPNs, like OpenVPN, thanks to the fact that it runs a module inside of the Linux kernel, it can use efficiently all CPU cores and it does not copy internal packets several times until they exit the network interface controller. All the interconnected clients are considered as peers. In my work there is one user (I will call it server) that offers the VPN service, through which all the traffic will be routed, and a second peer (called client) that wants to use this service.

3 Setup

For the tests I have used two machines, the first (the server) with Ubuntu 18.04, the second (the client) was running Ubuntu 20.04 on Virtual Box (the OS underneath was Windows 10).

3.1 Server

As first thing I have configured the server. I have installed the software using $apt - get\ install\ wireguard - dkms\ wireguard - tools\ linux - headers -$



```
[Interface]
Address = 10.100.100.1/24
SaveConfig = true
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT; iptables -t nat
-A POSTROUTING -o wlp1s0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT; iptables -t
nat -D POSTROUTING -o wlp1s0 -j MASQUERADE
ListenPort = 51820
PrivateKey = gIG9/rF0ov6Te+vwoYFtvC8PksakGAvD9iFLAeJEuV8=

[Peer]
PublicKey = FCJDRb66MHoAs78tSRFC2Lr9WdyzA4R24dV8+CtK0iU=
AllowedIPs = 10.100.100.2/32
Endpoint = 192.168.1.6:62625
```

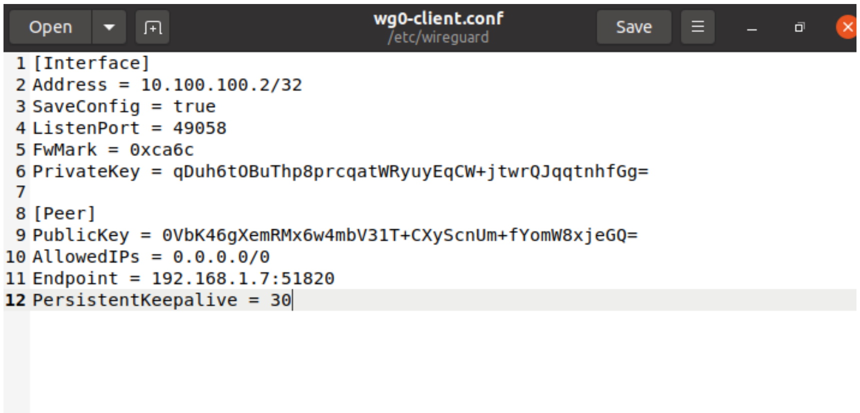
Figure 1: Configuration file for server

$\$(uname \ - r)$ and then I generated the private and public keys with the command:
wg genkey | *tee server_private_key* | *wg pubkey* > *server_public_key*.

At this point, the configuration file can be created at /etc/wireguard/wg0.conf and written as in image 1. ListenPort indicates the UDP port that has to be used (WireGuard uses only UDP), PrivateKey is the private key of the server, PublicKey refers to the client's public key and AllowedIPs represents the routing table that, in this case, is the address of the client (I just wanted these two users to communicate through the VPN in my network, but others can be added in the same way as the first one). To access the LAN I need to add PostUp and PostDown, they are needed to defined routing rules (NAT firewall rules and FORWARD rules). The last thing to set, server side, is the IPv4 forwarding, to let the client access the rest of the LAN and not only the server. This is done by opening the file /etc/sysctl.conf and uncommenting the line *net.ipv4.ip_forward=1*. To activate this last change, it is needed to restart the server or to use: *sysctl -p* and then *echo 1* | *sudo tee /proc/sys/net/ipv4/ip_forward*. So, after changing the permissions of the configuration file, the server can be start using *wg-quick up wg0* (or *systemctl enable wg-quick@wg0.service* to automatically initialize it every time the OS starts up). To stop the service use *wg-quick up wg0*.

3.2 Client

As has been done before for the server, the client needs to install the software, generate the keys using *wg genkey* and create a configuration file (image



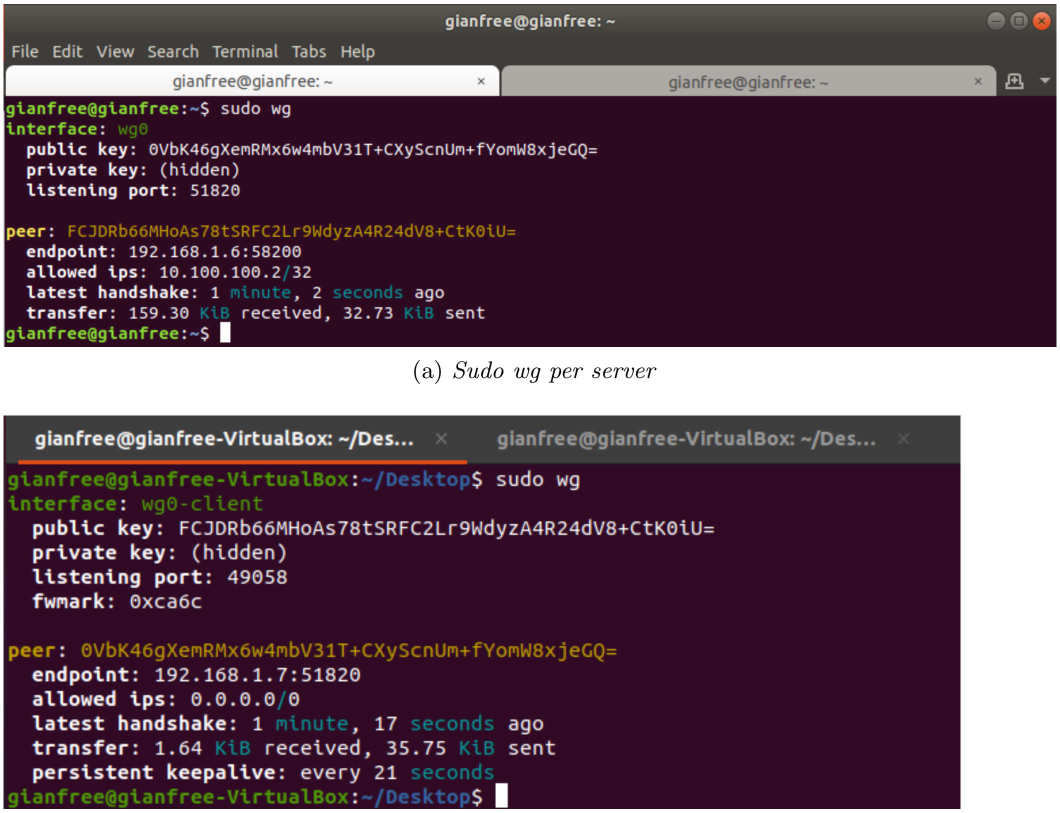
```
[Interface]
Address = 10.100.100.2/32
SaveConfig = true
ListenPort = 49058
FwMark = 0xca6c
PrivateKey = qDuh6tOBuThp8prcqatWRyuyEqCW+jtwrQJqqtnhfGg=

[Peer]
PublicKey = 0VbK46gXemRMx6w4mbV31T+CXyScnUm+fYomW8xjeGQ=
AllowedIPs = 0.0.0.0/0
Endpoint = 192.168.1.7:51820
PersistentKeepalive = 30
```

Figure 2: Configuration file for client

2). In this case we have AllowedIPs = 0.0.0.0/0 to route all the traffic from this peer through the server, Endpoint that identifies the server and PersistentKeepAlive that allows the peer to receive packets when he is not sending anyone and he is behind NAT or a firewall (i.e., every 30 seconds he will sends *keepalive packets* to keep NAT/Firewall mapping valid). Under the voice [Peer] we need server's info (Public key, IP address, UDP port). Now the client is ready to be activated (same way as for server).

4 Tests

After the setup, the public IP address for the client is equal to the server. Some informations about the peers connected can be shown using *wg* (in image 3). With the configurations I have reported before, I am able to ping the server from the client and vice versa (image 4), but, from the client, I was not able to ping other site, for example Google. The server receives the packets from the client (I used *tcpdump* to check it) but probably he was not able to send the answer back to the client. My problem was that in server's configuration file I wrote wrongly, in both PostUp and PostDown, the name of the network interface (1 is the revised version of the configuration file). Image 5 is a screenshot of the test on pinging google's DNS.



```
gianfree@gianfree:~$ sudo wg
interface: wg0
  public key: 0VbK46gXemRMx6w4mbV31T+CXyScnUm+fYomW8xjeGQ=
  private key: (hidden)
  listening port: 51820

peer: FCJDRb66MHoAs78tSRFC2Lr9WdyzA4R24dV8+CtK0iU=
  endpoint: 192.168.1.6:58200
  allowed ips: 10.100.100.2/32
  latest handshake: 1 minute, 2 seconds ago
  transfer: 159.30 KiB received, 32.73 KiB sent
gianfree@gianfree:~$
```

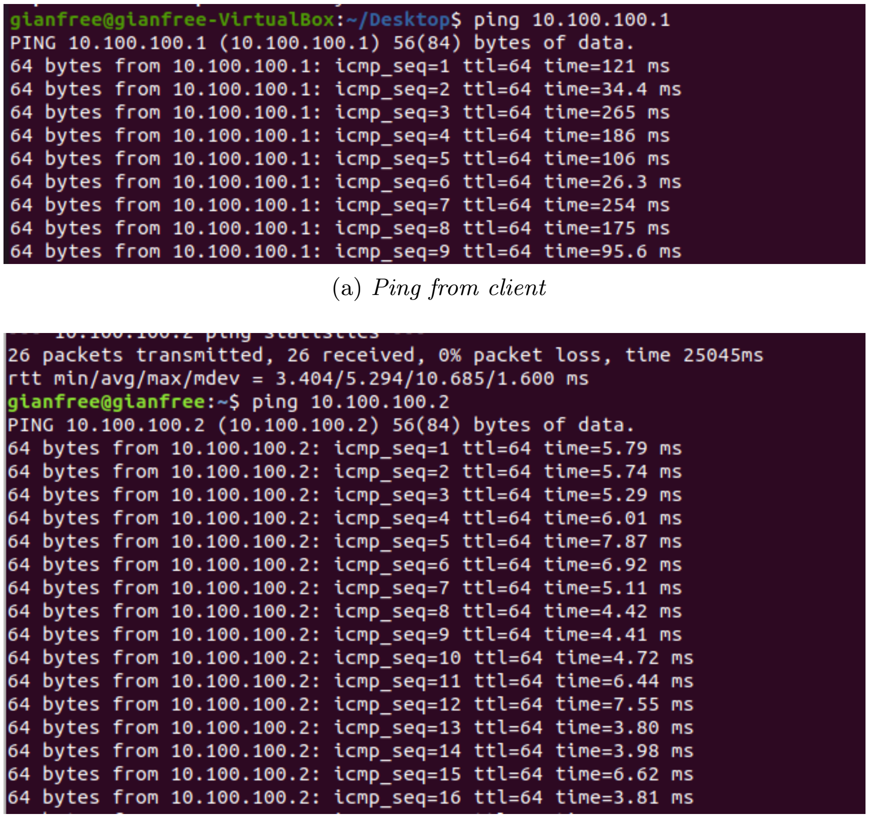
(a) *Sudo wg per server*

gianfree@gianfree-VirtualBox: ~/Des...
gianfree@gianfree-VirtualBox: ~/Des...

```
gianfree@gianfree-VirtualBox:~/Desktop$ sudo wg
interface: wg0-client
  public key: FCJDRb66MHoAs78tSRFC2Lr9WdyzA4R24dV8+CtK0iU=
  private key: (hidden)
  listening port: 49058
  fwmark: 0xca6c

peer: 0VbK46gXemRMx6w4mbV31T+CXyScnUm+fYomW8xjeGQ=
  endpoint: 192.168.1.7:51820
  allowed ips: 0.0.0.0/0
  latest handshake: 1 minute, 17 seconds ago
  transfer: 1.64 KiB received, 35.75 KiB sent
  persistent keepalive: every 21 seconds
gianfree@gianfree-VirtualBox:~/Desktop$
```

(b) *Sudo wg per client*

Figure 3: Info of the private network after setup



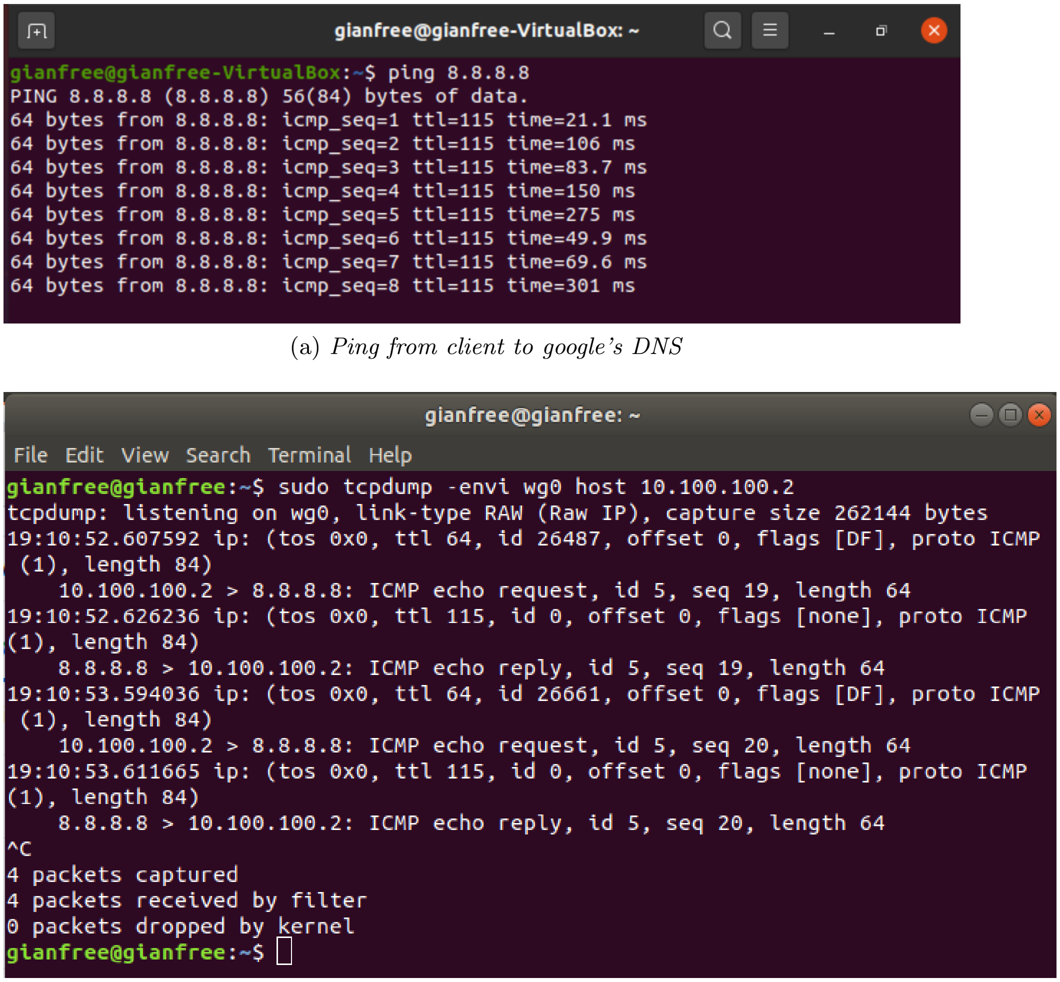
```
gianfree@gianfree-VirtualBox:~/Desktop$ ping 10.100.100.1
PING 10.100.100.1 (10.100.100.1) 56(84) bytes of data.
64 bytes from 10.100.100.1: icmp_seq=1 ttl=64 time=121 ms
64 bytes from 10.100.100.1: icmp_seq=2 ttl=64 time=34.4 ms
64 bytes from 10.100.100.1: icmp_seq=3 ttl=64 time=265 ms
64 bytes from 10.100.100.1: icmp_seq=4 ttl=64 time=186 ms
64 bytes from 10.100.100.1: icmp_seq=5 ttl=64 time=106 ms
64 bytes from 10.100.100.1: icmp_seq=6 ttl=64 time=26.3 ms
64 bytes from 10.100.100.1: icmp_seq=7 ttl=64 time=254 ms
64 bytes from 10.100.100.1: icmp_seq=8 ttl=64 time=175 ms
64 bytes from 10.100.100.1: icmp_seq=9 ttl=64 time=95.6 ms
```

(a) *Ping from client*

```
--- 10.100.100.2 ping statistics ---
26 packets transmitted, 26 received, 0% packet loss, time 25045ms
rtt min/avg/max/mdev = 3.404/5.294/10.685/1.600 ms
gianfree@gianfree:~$ ping 10.100.100.2
PING 10.100.100.2 (10.100.100.2) 56(84) bytes of data.
64 bytes from 10.100.100.2: icmp_seq=1 ttl=64 time=5.79 ms
64 bytes from 10.100.100.2: icmp_seq=2 ttl=64 time=5.74 ms
64 bytes from 10.100.100.2: icmp_seq=3 ttl=64 time=5.29 ms
64 bytes from 10.100.100.2: icmp_seq=4 ttl=64 time=6.01 ms
64 bytes from 10.100.100.2: icmp_seq=5 ttl=64 time=7.87 ms
64 bytes from 10.100.100.2: icmp_seq=6 ttl=64 time=6.92 ms
64 bytes from 10.100.100.2: icmp_seq=7 ttl=64 time=5.11 ms
64 bytes from 10.100.100.2: icmp_seq=8 ttl=64 time=4.42 ms
64 bytes from 10.100.100.2: icmp_seq=9 ttl=64 time=4.41 ms
64 bytes from 10.100.100.2: icmp_seq=10 ttl=64 time=4.72 ms
64 bytes from 10.100.100.2: icmp_seq=11 ttl=64 time=6.44 ms
64 bytes from 10.100.100.2: icmp_seq=12 ttl=64 time=7.55 ms
64 bytes from 10.100.100.2: icmp_seq=13 ttl=64 time=3.80 ms
64 bytes from 10.100.100.2: icmp_seq=14 ttl=64 time=3.98 ms
64 bytes from 10.100.100.2: icmp_seq=15 ttl=64 time=6.62 ms
64 bytes from 10.100.100.2: icmp_seq=16 ttl=64 time=3.81 ms
```

(b) *Ping from server*

Figure 4: Packets sent between the peers



```
gianfree@gianfree-VirtualBox:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=21.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=106 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=83.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=150 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=275 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=49.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=69.6 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=301 ms
```

(a) *Ping from client to google's DNS*

```
gianfree@gianfree:~$ sudo tcpdump -envi wg0 host 10.100.100.2
tcpdump: listening on wg0, link-type RAW (Raw IP), capture size 262144 bytes
19:10:52.607592 ip: (tos 0x0, ttl 64, id 26487, offset 0, flags [DF], proto ICMP
 (1), length 84)
    10.100.100.2 > 8.8.8.8: ICMP echo request, id 5, seq 19, length 64
19:10:52.626236 ip: (tos 0x0, ttl 115, id 0, offset 0, flags [none], proto ICMP
(1), length 84)
    8.8.8.8 > 10.100.100.2: ICMP echo reply, id 5, seq 19, length 64
19:10:53.594036 ip: (tos 0x0, ttl 64, id 26661, offset 0, flags [DF], proto ICMP
 (1), length 84)
    10.100.100.2 > 8.8.8.8: ICMP echo request, id 5, seq 20, length 64
19:10:53.611665 ip: (tos 0x0, ttl 115, id 0, offset 0, flags [none], proto ICMP
(1), length 84)
    8.8.8.8 > 10.100.100.2: ICMP echo reply, id 5, seq 20, length 64
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
gianfree@gianfree:~$
```

(b) *tcpdump in server*

Figure 5: Ping of the client to Google passes through the server

References

[1] https://www.wireguard.com/quickstart/

[2] https://it.wikipedia.org/wiki/WireGuard