VPN Setup & Secure Remote Access Lab

Introduction

Virtual private networks (VPNs) are used to establish secure, encrypted connections between devices over untrusted networks. By encapsulating traffic inside an encrypted tunnel, a VPN ensures that data cannot be intercepted or tampered with by third parties. In this lab you will build on previous hardening work to set up a secure remote-access VPN using **WireGuard**, a modern tunnelling protocol known for its simplicity and performance.

Objective and Scope

The objective of this exercise is to implement and verify a basic WireGuard VPN that allows a client to connect securely to a hardened Linux server. The focus is on understanding key generation, configuration of server and client, tunnel establishment, and basic connectivity tests. The lab environment consists of two machines:

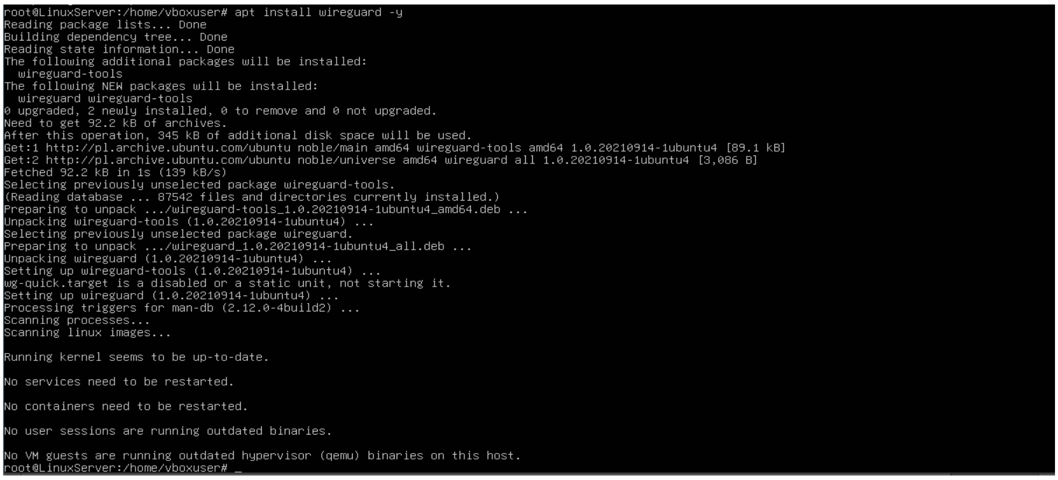
- **Client** – A physical workstation running Windows 11. This machine acts as the remote user that needs secure access to the server.
- **Server** – A virtual machine (Ubuntu Server 24.04) running under Oracle VirtualBox. The VM uses the same distribution version and network configuration (bridged adapter) as the earlier server hardening exercise.

During the exercise, you will generate cryptographic keys, configure WireGuard interfaces on both systems, open necessary firewall ports, and validate the VPN connection with basic network tests. Screenshots captured throughout these steps will serve as evidence in the final report.

1. Installing WireGuard

Server (Ubuntu)

To prepare the Linux server, install the wireguard package.

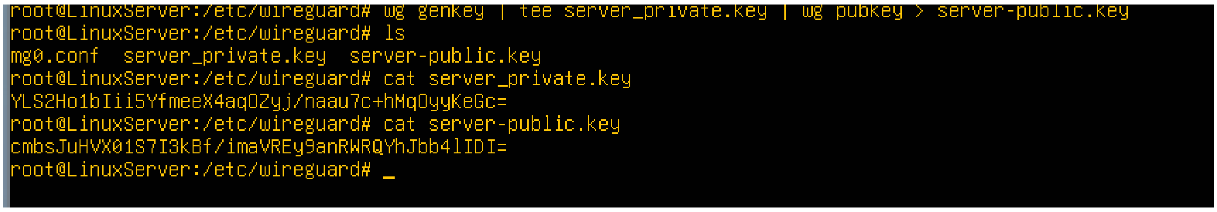


```
root@LinuxServer:/home/vboxuser# apt install wireguard -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  wireguard-tools
The following NEW packages will be installed:
  wireguard wireguard-tools
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 92.2 kB of archives.
After this operation, 345 kB of additional disk space will be used.
Get:1 http://pl.archive.ubuntu.com/ubuntu noble/main amd64 wireguard-tools amd64 1.0.20210914-1ubuntu4 [89.1 kB]
Get:2 http://pl.archive.ubuntu.com/ubuntu noble/universe amd64 wireguard all 1.0.20210914-1ubuntu4 [3,086 B]
Fetched 92.2 kB in 1s (139 kB/s)
Selecting previously unselected package wireguard-tools.
(Reading database ... 87542 files and directories currently installed.)
Preparing to unpack .../wireguard-tools_1.0.20210914-1ubuntu4_amd64.deb ...
Unpacking wireguard-tools (1.0.20210914-1ubuntu4) ...
Selecting previously unselected package wireguard.
Preparing to unpack .../wireguard_1.0.20210914-1ubuntu4_all.deb ...
Unpacking wireguard (1.0.20210914-1ubuntu4) ...
Setting up wireguard-tools (1.0.20210914-1ubuntu4) ...
wg-quick.target is a disabled or a static unit, not starting it.
Setting up wireguard (1.0.20210914-1ubuntu4) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@LinuxServer:/home/vboxuser# _
```

2.WireGuard uses a simple key-pair mechanism. Each side (server and client) requires its own private and public key pair.

Server

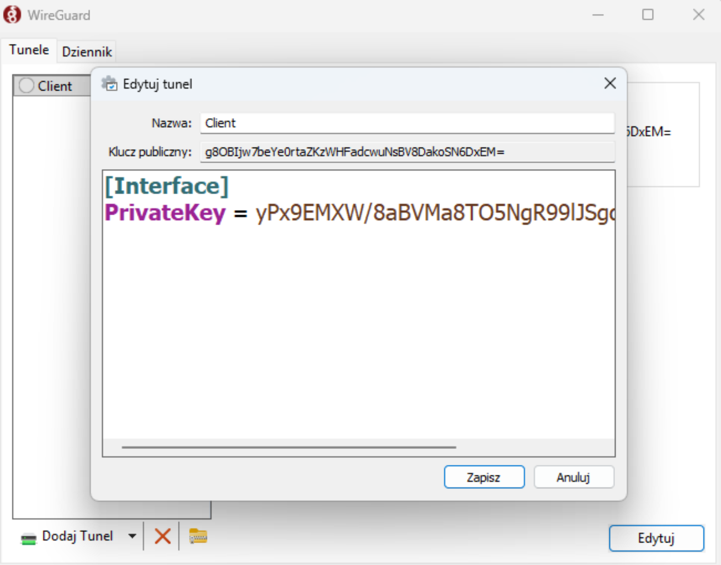
Generate the server keys and save them to files. The private key is kept secret, while the public key is shared with peers.



```
root@LinuxServer:/etc/wireguard# wg genkey | tee server_private.key | wg pubkey > server-public.key
root@LinuxServer:/etc/wireguard# ls
mg0.conf  server_private.key  server-public.key
root@LinuxServer:/etc/wireguard# cat server_private.key
YLS2Ho1bIii5YfmeeX4aqOZyj/naau7c+hMqOyyKeGc=
root@LinuxServer:/etc/wireguard# cat server-public.key
cmbsJuHVX01S7I3kBf/imaVREy9anRWRQYhJbb4lIDI=
root@LinuxServer:/etc/wireguard# _
```

Client

On Windows, open the WireGuard application and choose **Add empty tunnel**. The client will automatically generate a pair of keys. Record the **PrivateKey** for the [Interface] section and the **PublicKey** for the server configuration.



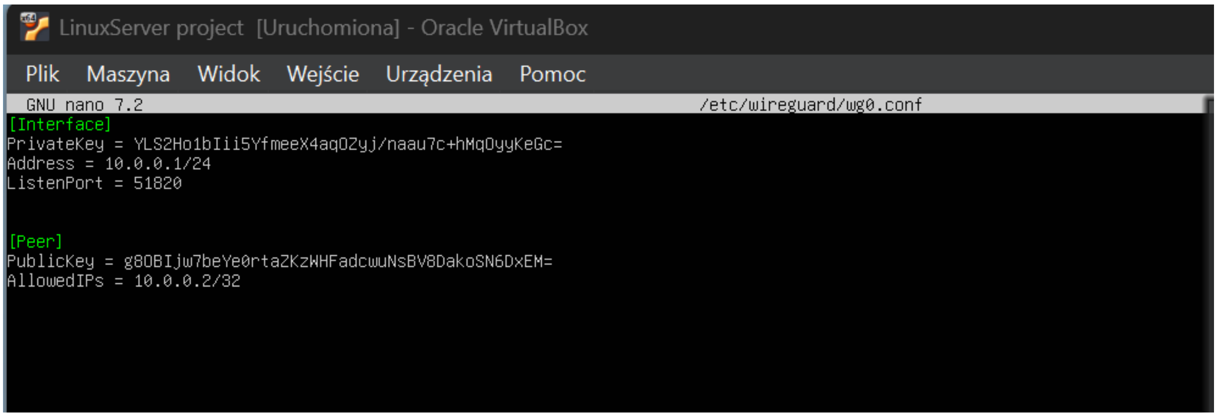
Nazwa: Client

Klucz publiczny: g8OBIjw7beYe0rtaZKzWHFadcwuNsBV8DakoSN6DxEM=

```
[Interface]
PrivateKey = yPx9EMXW/8aBVMa8TO5NgR99lJSgc
```

3. Server Configuration

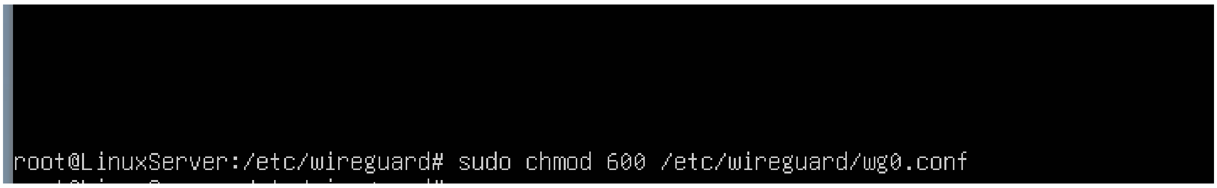
Create the WireGuard configuration file on the server at /etc/wireguard/wg0.conf. Fill it with the server’s private key, assign an internal VPN address, and define the client peer using its public key and allowed IP.



```
  GNU nano 7.2                                                   /etc/wireguard/wg0.conf
[Interface]
PrivateKey = YLS2Ho1bIii5YfmeeX4aqOZyj/naau7c+hMqOyyKeGc=
Address = 10.0.0.1/24
ListenPort = 51820


[Peer]
PublicKey = g8OBIjw7beYe0rtaZKzWHFadcwuNsBV8DakoSN6DxEM=
AllowedIPs = 10.0.0.2/32
```

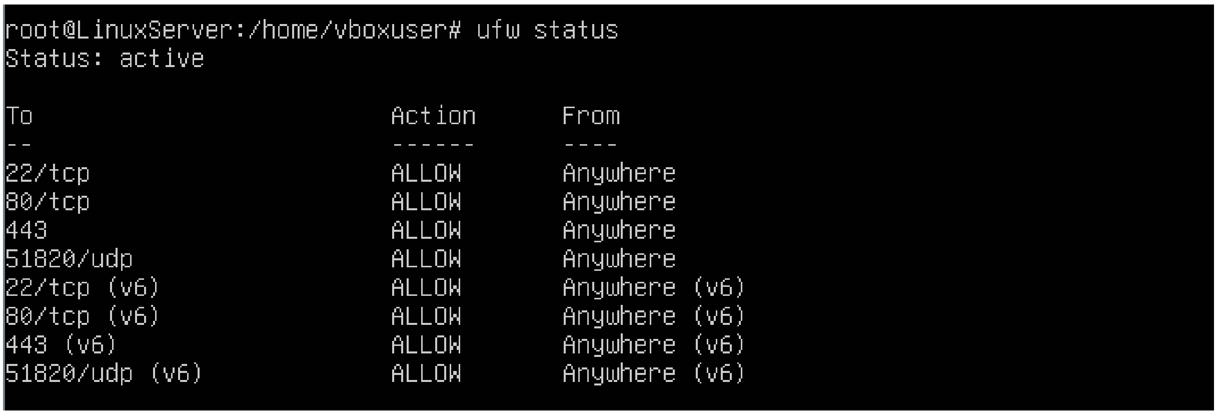
- **PrivateKey** – value from server_private.key.
- **Address** – the VPN IP address for the server within the 10.0.0.0/24 subnet.
- **ListenPort** – port on which WireGuard listens (UDP 51820).
- **PublicKey** – the client's public key from the Windows app.
- **AllowedIPs** – the exact VPN IP assigned to the client.

Save the file and secure it so only root can read it:



```
root@LinuxServer:/etc/wireguard# sudo chmod 600 /etc/wireguard/wg0.conf
```

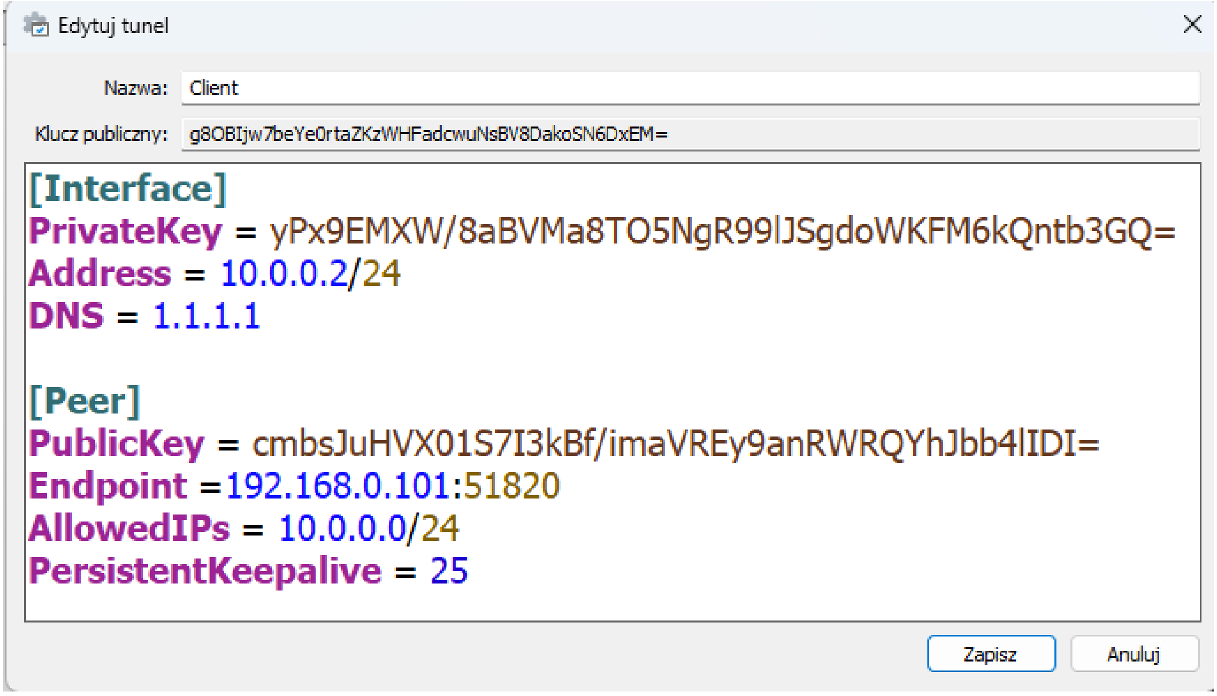
Allow the WireGuard port through the firewall:



```
root@LinuxServer:/home/vboxuser# ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443                        ALLOW       Anywhere
51820/udp                  ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
51820/udp (v6)             ALLOW       Anywhere (v6)
```

4. Client Configuration

Configure the Windows client using the keys and server details. In the WireGuard GUI, click **Add empty tunnel** and fill in the fields:



Nazwa: Client

Klucz publiczny: g8OBIjw7beYe0rtaZKzWHFadcwuNsBV8DakoSN6DxEM=

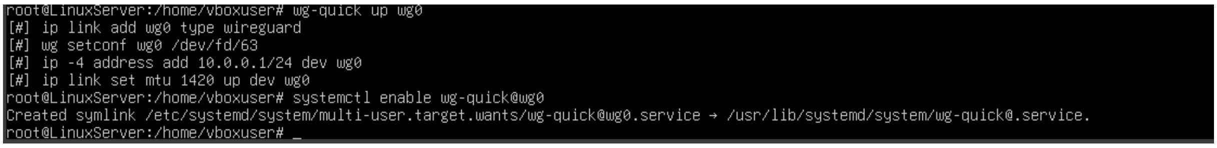
```
[Interface]
PrivateKey = yPx9EMXW/8aBVMa8TO5NgR99lJSgdoWKFM6kQntb3GQ=
Address = 10.0.0.2/24
DNS = 1.1.1.1

[Peer]
PublicKey = cmbsJuHVX01S7I3kBf/imaVREy9anRWRQYhJbb4lIDI=
Endpoint =192.168.0.101:51820
AllowedIPs = 10.0.0.0/24
PersistentKeepalive = 25
```

- **PrivateKey** – the client's private key generated earlier.
- **Address** – the VPN IP for the client.
- **DNS** – optional; sets DNS resolver used inside the tunnel.
- **PublicKey** – the server's public key from server_public.key.
- **Endpoint** – the server's LAN IP address and listening port (e.g., 192.168.0.104:51820).
- **AllowedIPs** – which destination IPs go through the tunnel; here the entire VPN subnet.
- **PersistentKeepalive** – sends periodic keepalive packets to maintain NAT mappings (25 seconds is typical)

5. Starting the Tunnel

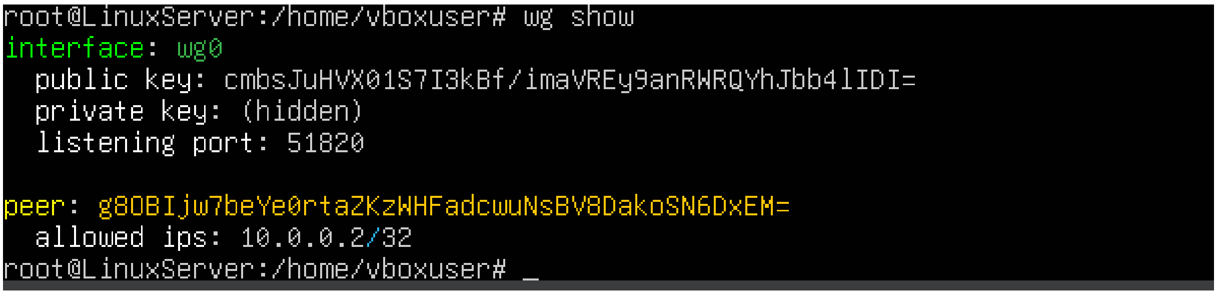
Server

Bring up the WireGuard interface using the helper script and enable automatic startup on boot.



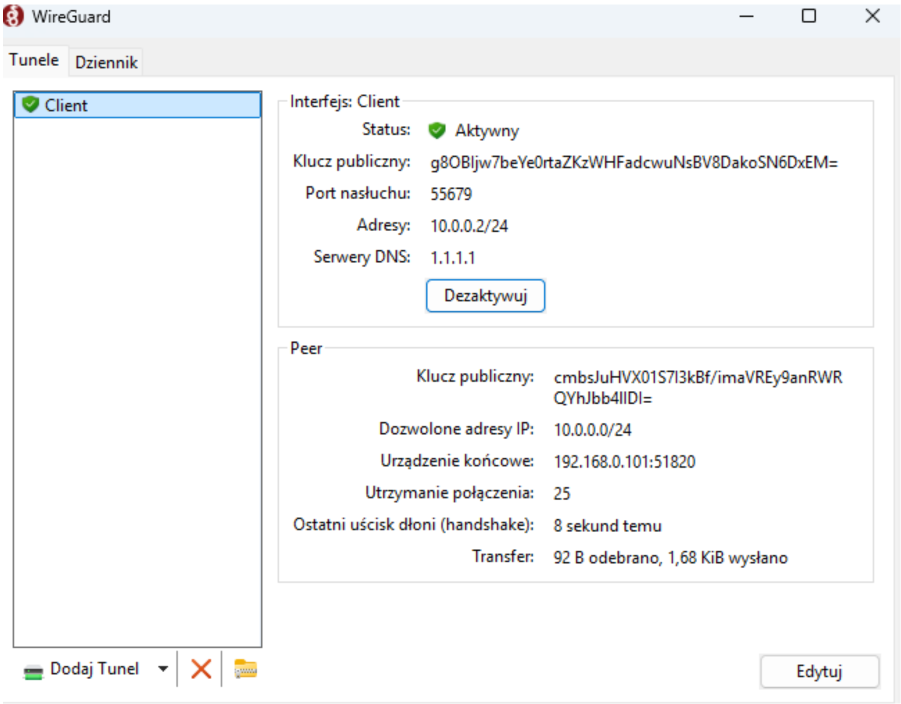
```
root@LinuxServer:/home/vboxuser# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
root@LinuxServer:/home/vboxuser# systemctl enable wg-quick@wg0
Created symlink /etc/systemd/system/multi-user.target.wants/wg-quick@wg0.service → /usr/lib/systemd/system/wg-quick@.service.
root@LinuxServer:/home/vboxuser# _
```

Verify that the interface is running and waiting for peers:



```
root@LinuxServer:/home/vboxuser# wg show
interface: wg0
  public key: cmbsJuHVX01S7I3kBf/imaVREy9anRWRQYhJbb4lIDI=
  private key: (hidden)
  listening port: 51820

peer: g8OBIjw7beYe0rtaZKzWHFadcwuNsBV8DakoSN6DxEM=
  allowed ips: 10.0.0.2/32
root@LinuxServer:/home/vboxuser# _
```

At this point, wg show will display the server's public key and listening port. The **latest handshake** and **transfer** fields will be blank until the client connects.



Interfejs: Client

Status: Aktywny
Klucz publiczny: g8OBIjw7beYe0rtaZKzWHFadcwuNsBV8DakoSN6DxEM=
Port nasłuchu: 55679
Adresy: 10.0.0.2/24
Serwery DNS: 1.1.1.1

Peer

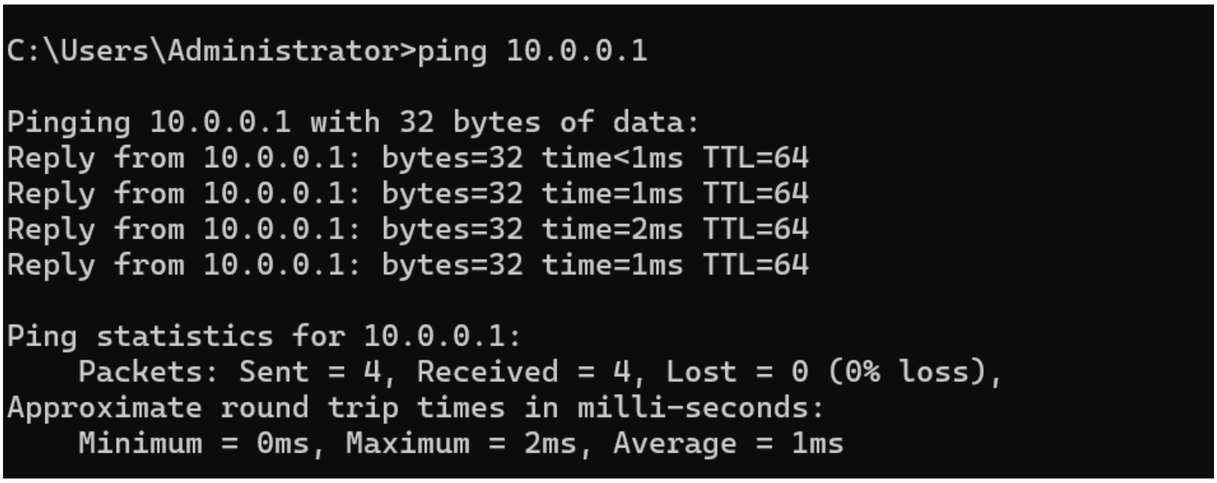
Klucz publiczny: cmbsJuHVX01S7I3kBf/imaVREy9anRWRQYhJbb4lIDI=
Dozwolone adresy IP: 10.0.0.0/24
Urządzenie końcowe: 192.168.0.101:51820
Utrzymanie połączenia: 25
Ostatni uścisk dłoni (handshake): 8 sekund temu
Transfer: 92 B odebrano, 1,68 KiB wysłano

6. Testing Connectivity

After the tunnel is established, verify that traffic flows through it.

1. **Ping from client to server**:

On Windows, open a command prompt and run:

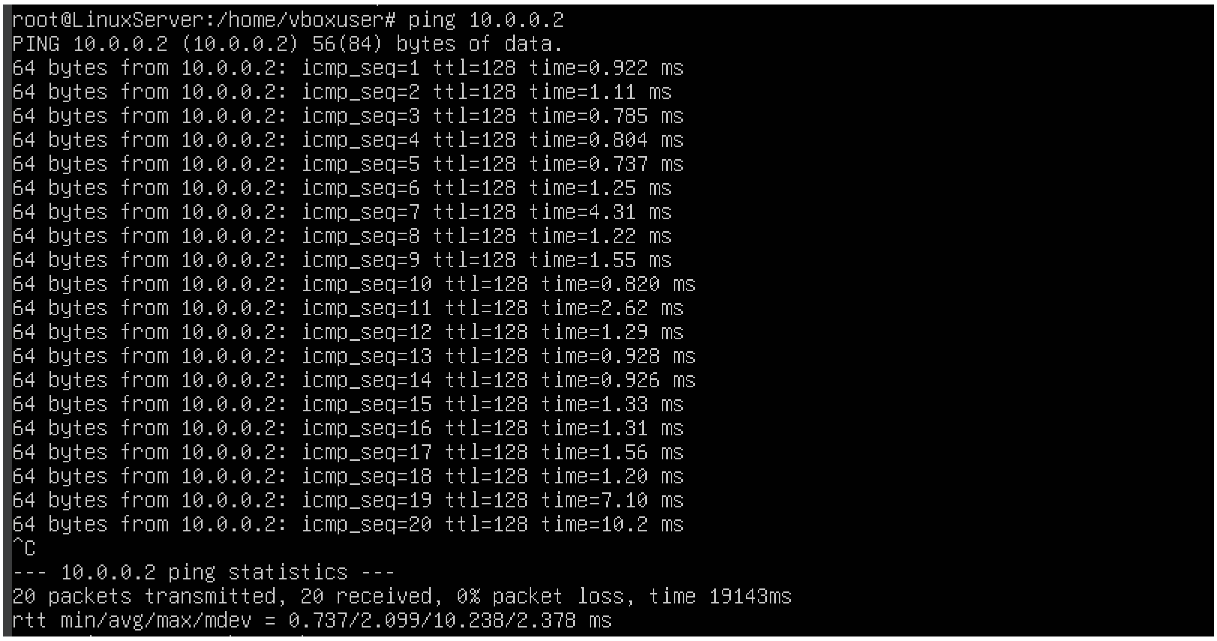


```
C:\Users\Administrator>ping 10.0.0.1

Pinging 10.0.0.1 with 32 bytes of data:
Reply from 10.0.0.1: bytes=32 time<1ms TTL=64
Reply from 10.0.0.1: bytes=32 time=1ms TTL=64
Reply from 10.0.0.1: bytes=32 time=2ms TTL=64
Reply from 10.0.0.1: bytes=32 time=1ms TTL=64

Ping statistics for 10.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 2ms, Average = 1ms
```

You should receive replies from the server's VPN IP.

2 . **Ping from server to client**:

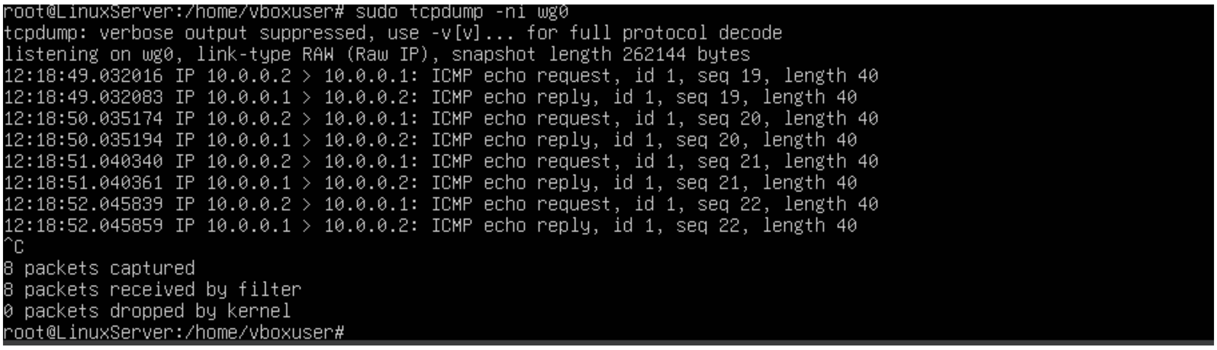
On the server:



```
root@LinuxServer:/home/vboxuser# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=128 time=0.922 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=128 time=1.11 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=128 time=0.785 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=128 time=0.804 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=128 time=0.737 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=128 time=1.25 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=128 time=4.31 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=128 time=1.22 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=128 time=1.55 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=128 time=0.820 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=128 time=2.62 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=128 time=1.29 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=128 time=0.928 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=128 time=0.926 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=128 time=1.33 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=128 time=1.31 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=128 time=1.56 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=128 time=1.20 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=128 time=7.10 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=128 time=10.2 ms
^C
--- 10.0.0.2 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19143ms
rtt min/avg/max/mdev = 0.737/2.099/10.238/2.378 ms
```

Capture tunnel traffic(I used tcpdump)

On the server, monitor ICMP packets on the wg0 interface to see the VPN traffic:



```
root@LinuxServer:/home/vboxuser# sudo tcpdump -ni wg0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on wg0, link-type RAW (Raw IP), snapshot length 262144 bytes
12:18:49.032016 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 1, seq 19, length 40
12:18:49.032083 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 1, seq 19, length 40
12:18:50.035174 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 1, seq 20, length 40
12:18:50.035194 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 1, seq 20, length 40
12:18:51.040340 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 1, seq 21, length 40
12:18:51.040361 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 1, seq 21, length 40
12:18:52.045839 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 1, seq 22, length 40
12:18:52.045859 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 1, seq 22, length 40
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
root@LinuxServer:/home/vboxuser#
```

Conclusion

This lab demonstrated how to configure a simple WireGuard VPN for secure remote access. By generating cryptographic keys, defining server and client configurations, starting the tunnel, and validating connectivity with ping and packet captures, you created a functional encrypted link between a Windows 11 client and a hardened Ubuntu server.

Author: Maciej Łęczycki

Date: 14.09.2025