Virtual Private Networking

Wireguard

I had two goals I wanted to accomplish with VPNs.

1. I need to redirect all outbound traffic from a specific subnet through a VPN so any client on that subnet would have its privacy protected by the VPN.
2. Allow me to VPN into my home network from somehwere else and have access to everything as if I was sitting on my computer at home.

Both of them could have been accomlished with any VPN most likely but I went with WireGuard since it is a simple and fast VPN whose setup is similar to SSH so it was inuitive for me to setup.

Host Setup

To use *Wireguard* inside of a LXC container the host has to have *Wireguard* installed since *LXC* guests are run with the kernel of the host system. *Wireguard* was first mainlined into the *Linux* kernel in version 5.6 so with kernel versions using 5.6 or later it is already built in. Anything before 5.6 that doesn't specifically have *Wireguard* backported in will need to use kernel modules to get it working. Wireguard.com has detailed instructions on how to install it on nearly any platform. Since I am using *Proxmox* as my host it was already backported into the kernel.

Guest Setup

Additionally I needed the `wireguard-tools` package in the *LXC* guest and `resolvconf` so DNS can be configured properly.

```
# echo "deb http://deb.debian.org/debian buster-backports main" > /etc/apt/sources.list.d/buster-backports.list
# apt update
# apt install --no-install-recommends wireguard-tools
# apt install resolvconf
```

Route Subnet Through Wireguard Interface

Funneling all traffic from an Ethernet interface through a *Wireguard* interface is relatively easy once I became familar with how packets flow through *Linux*. I mostly just needed to modify my *Wireguard* `*.conf` file to add the `Table`, `PostUp`, and `PreDown` parameters.

> I also needed to setup IP masquerading of outgoing traffic on my *Wireguard* interface. See below for instructions.

Create Interface

Creating the configuration file is a bit out of the scope of this document. A VPN provider that supports *Wireguard* will likely just provide a pre-built configuration file. But I also have a brief overview of how you'd make one.

```
# /etc/wireguard/warp.conf
[Interface]
PrivateKey = ****
Address = 10.10.20.59/19, 2a03:4012:4021:80af::1f3c/64
DNS = 10.10.0.1, 2a03:4012:4021:80af::1
Table = 9
PostUp = ip rule add iif eth1.9 lookup 9; ip -6 rule add iif eth1.9 lookup 9
PreDown = ip rule del iif eth1.9 lookup 9; ip -6 rule del iif eth1.9 lookup 9

[Peer]
PublicKey = T28Qn5VFzT4wiwEPd7DscwcP3Rsmq23QcnjH1N5G/wc=
Endpoint = wireguard.vpn-provider.example:51820
AllowedIPs = 0.0.0.0/0, ::0/0...
```

Line 5: All rules/routes should be applied to a custom route table `9`. I could have also named my custom route table by running `echo "9 warp" > /etc/iproute2/rt_tables` and then say `Table = warp` for improved readability.

Line 6: Adds rules for IPv4 and IPv6 that all traffic coming in interface `eth1.9` should use custom route table `9`. Because I defined a peer with `AllowedIPs = 0.0.0.0/0, ::0/0` a default route will be setup on custom route table `9` that redirects all traffic to the *Wireguard* interface. If I named my custom route like shown above I could have said `lookup warp` inplace of `lookup 9`.

Line 7: Just the inverse of line 5 to clean up after myself when taking down the *Wireguard* interface.

Setup IP Masquerading

> IP Masquerading is a technique that hides an entire IP address space, usually consisting of private IP addresses, behind a single IP address in another, usually public address space.

Source: Wikipedia

Configuration

The easiest way to set this up are to append some *netfilter* rules to the `PostUp` and `PreDown` parameters.

```
...
PostUp = ...; iptables -t nat -A POSTROUTING -o wg0 -j MASQUERADE
PreDown = ...; iptables -t nat -D POSTROUTING -o wg0 -j MASQUERADE


[Peer]
...
```

Although this works fine there is a risk of the *iptables/netfilter* rules getting squashed by *Shorewall* if it is restarted while the *Wireguard* interface exists. It is best to have *Shorewall* setup the masquerading by making a simple declaration in `/etc/shorewall/snat`. I've included the other *Shorewall* configuration files that would be necessary to make this setup work.

First I define the `wg` zone…

```
# /etc/shorewall/zones
 #ZONE   TYPE    OPTIONS                IN                  OUT
 #                               OPTIONS             OPTIONS
```

```
  warp    ipv4
+ wg      ipv4
```

Then I define the interface `WG_IF` and put it in the `wg` zone…

```
# /etc/shorewall/interfaces
  #ZONE	INTERFACE	OPTIONS
  warp	WARP_IF		tcpflags,nosmurfs,routefilter=2,logmartians,physical=eth1.9
+ wg	WG_IF		physical=wg0
```

This tells Shorwall to masquerade all IPs going out on `WG_IF`…

```
# /etc/shorewall/snat
  #ACTION		SOURCE		DEST
+ MASQUERADE	0.0.0.0/0	WG_IF
```

Then I allow the `warp` zone to send packets to the `wg` zone. The `warp` zone isn't allowed to send packets to any other subnet or the `wan`. This prevents any data/privacy spills from happening if the *Wireguard* interface ever goes down. It is always best to fail into a state that protects security and privacy.

```
# /etc/shorewall/policy
  #SOURCE	DEST		POLICY          LOGLEVEL        RATE    CONNLIMIT
- warp		$FW			ACCEPT          $LOG_LEVEL
+ warp		$FW,wg		ACCEPT          $LOG_LEVEL
```

Remote Access

Allowing remote access is just a matter of setting up a new *Wireguard* interface, allowing incoming traffic to that interface, and making sure the firewall allows that traffic to connect to the rest of the network.

Create Interface

```
# cd /etc/wireguard
# umask 077
# wg genkey | tee guard.key | wg pubkey > guard.pub
# printf "[Interface]\PrivateKey = %s\n" `cat guard.key`
```

Then I modified my file to finish configuring the interface and allow a `[Peer]` for my laptop.

```
# /etc/wireguard/guard.conf
[Interface]
PrivateKey = ****
+ Address = 10.0.2.1/28, 2001:db8:2ebf:2::1/64
+ ListenPort = 51820
+
+ [Peer]
+ PublicKey = Iz5ceR0+tCN3BLTWehZxSplzdbABRT8geqifFxubHUA=
+ AllowedIPs = 10.0.2.4/32, 2001:db8:2ebf:1::4/128
+ PresharedKey = ***
```

Line 4: Sets an IPv4 and IPv6 address for this interface. These will be the servers IPs on each virtual subnet.

Line 5: Sets the port to listen to for this interface. It is just the default *Wirgaurd* port and I'll allow traffic through the firewall for it soon.

Line 7-10: Declare a peer, define the public key to use when communicating and validaing any connections, set what IPs the peer is allowed to use on each virtual subnet, and configure a pre-shared key for additional secuirty.

> A preshard key can be generated by running `wg genpsk` and must be the same on both the `[Peer]` block on the server and the `[Interface]` block on the client.

Firewall Configuration

First I had to declare a new interface and since I want it to be as if I was sitting on my laptop at home, I put it in the `lan` zone.

```
# /etc/shorewall/interfaces
...
 #ZONE	INTERFACE	OPTIONS
...
 wg	WGAZSE1_IF	tcpflags,nosmurfs,routefilter,logmartians,physical=wgazse1
+ lan	WGGUARD_IF	tcpflags,nosmurfs,routefilter,logmartians,physical=wgguard
```

```
# /etc/shorewall/interfaces
...
 #ZONE	INTERFACE	OPTIONS
...
 wg	WGAZSE1_IF	tcpflags,nosmurfs,routefilter,logmartians,physical=wgazse1
+ lan	WGGUARD_IF	tcpflags,forward=1,physical=wgguard
```

For outside clients to connect I need to add a rule that allows them to connect to the firewall on port 51820.

```
# /etc/shorewall[6]/rules
+ ACCEPT        wan,lan        $FW          udp          51820
```

The last step is to once again setup masquerading so traffic from clients on the *Wireguard* subnet appear to be originating from the `wgguard` interface which is in the `lan` zone.

```
# /etc/shorewall/snat
+ MASQUERADE		10.0.2.0/28				WAN_IF,LAN_IF,DMZ_IF
```

```
# /etc/shorewall6/snat
+ MASQUERADE		fde9:2375:2ebf:2::/64	WAN_IF,LAN_IF,DMZ_IF
```