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[]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\$FWACCEPT \$LOG_LEVEL			
+ warp <u></u> \$FW,wg <u></u> ACCEPT	\$LOG_LEVEL		

Revision #5 Created 2 April 2021 02:44:17 by dustin@swigg.net Updated 8 April 2021 13:07:48 by dustin@swigg.net