Birdc

WireGuard is generally described on another page, here: VPN - Wireguard. This page is about what is needed to configure WireGuard for routing over the VPN; especially with a focus on OSPF.

A Note on Cryptokey Routing

It's worth a section to touch on the cryptokey routing feature of WireGuard and how it works with the mesh.

All WireGuard nodes list their peers in a configuration file. Among the peer configuration is a public key and a list of acceptable IP ranges for the peer. Once the tunnel is brought up, packets from inside the tunnel must match the IPs in the list. Packets are routed to the peer using the IP range list and encrypted for the destination peer with its specific public key. Given this, no two peers may have overlapping IP ranges. Therefore, routing through two different peers to another peer downstream, or the internet, on a single wireguard connection cannot be accomplished using WireGuard in this manner.

However, the cryptokey routing is *per-interface*. It's possible for an interface to allow "all IPs" (`0.0.0.0/0`) to/from a peer. All IPs and dynamic routing can be accomplished over a fully open WireGuard interface, but only with one other peer, and one new interface for each peer pair. Therefore for routed WireGuard connections a special configuration is required on both ends to make this possible.

Process

Each side of a routed WireGuard VPN link will need the following:

- A dedicated linux server to be the VPN router.
- A new WireGuard interface for the other side of the VPN.
- A routing daemon, probably BIRD, which can perform OSPF in PtP mode.

Dedicate Linux Server

The server on each side of the VPN will need to be configured appropriately. When handling the VPN, the server will have "two internets", and possibly other traffic direction. As such, it's important

to configure the system correctly so that it does not "leak" traffic from the mesh onto the internet, vice versa, nor from any other network the same server handles. It is best to use a dedicated box, which will still need some custom "rules" to make it happen.

There are a few approaches to handle the traffic separation:

1) Multiple Routing Tables and firewalling

Linux has the ability to handle multiple routing tables. They are easy to define through a single file in the `/etc/` path. The BIRD routing daemon can also be configured to manage routes in the non-standard routing tables, so these two pieces easily work together.

One challenge with multiple routing tables is that a single device (WireGuard endpoint or Ethernet card) cannot be "tied" to a routing table. Instead, a rule (in addition to the firewall) is needed, to select the the routing table used for making a decision. These rules are easy to create but can be somewhat challenging to get right, and there is no good universal way to manage them.

Set up a new routing table for the mesh:

Edit `/etc/iproute2/rt_tables` and add a second table for the mesh:

```
$ vi /etc/iproute2/rt_tables
1 admin
2 nycmesh
3 public
```

Modify network file to add ip rules

Later in this write-up will describe how to add WireGuard interfaces and create their configuration in (if using Debian/Ubuntu) `/etc/network/interfaces.d/*`. If using the multiple routing tables method, additional lines will need to be added. They will be added in the example below, but show specificially here:

```
auto wg111
iface wg111 inet static
        address 10.70.xx.1/31
        pre-up ip link add $IFACE type wireguard
        pre-up wg setconf $IFACE /etc/wireguard/$IFACE.conf
        pre-up ip link set up dev $IFACE
        pre-up ip route add 10.70.xx.1/31 dev $IFACE table nycmesh
        pre-up ip rule add iif $IFACE pref 1031 table nycmesh
        pre-up ip rule add from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
```

```
        post-down ip rule del iif $IFACE table nycmesh
        post-down ip rule del from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
```

Note the order of the `pre-up` and `post-down` rules, as it is critical. Some systems may vary in terms of the rules, depending on your distribution. Report back and help us update the document if something is off.

Note also the `table nycmesh` which is referenced by name. The name needs to match the `rt_tables` name.

Local Interfaces, to mesh-routers at the same site, will also need ip rules. For example:

```
# Mesh internal
allow-hotplug <Local Interface>
iface <Local Interface> inet static
        address 10.69.XX.YY/16
        post-up ip route add 10.69.0.0/16 dev $IFACE table nycmesh
        post-up ip rule add iif $IFACE pref 1021 table nycmesh
        post-up ip rule add from 10.69.XX.YY table nycmesh
        post-down ip route del 10.69.0.0/16 dev $IFACE table nycmesh
        post-down ip rule del iif $IFACE table nycmesh
        post-down ip rule del from 10.69.XX.YY table nycmesh
```

2) Linux Namespaces / Containers.

Alternately, a newer Linux feature, namespaces, which is what containers are based on, can be used to provide segregation. Although containers and namespaces are intertwined, containers have a highly-simplified version of the general network namespace concept. As such, containers are not directly suitable for use as routing namespaces without much extra work. Therefore, the more challenging raw namespaces much be used.

This way has not yet been explored, it is more theoretical, so follow the first approach for now

WireGuard interface for each side

Each side's dedicated Linux server will need a *new* WireGuard interface for the connection. This is because each side needs to use `0.0.0.0/0` as the AllowedIPs so that any mesh address can be routed over the connection.

The connection will also need a pair of unique private IP addresses to route the internal traffic. *At this time, the addresses should be taken from the NYC Mesh IP Ranges spreadsheet to ensure they are unique. Please ask Zach for an address pair.*

Below are WireGuard configuration files which can be used as a basis for setting up a connect. (Be sure to replace the keys and addresses with the proper inputs).

Note the suggested device names -- It is recommended to name the interface as `wgX` where `X` is the destination node number the connection. This makes troubleshooting and configuration much easier..

Side 1 (Node 111):

```
# This is Node 111 Interface wg222
[Interface]
PrivateKey = ThisIsThePrivateKeyOnSide1
ListenPort = 51825

# Node 222
[Peer]
PublicKey = ThisIsThePublicKeyFromSide2
AllowedIPs = 0.0.0.0/0
```

Side 2 (Node 222):

```
# This is Node 222 Interface wg111
[Interface]
PrivateKey = ThisIsThePrivateKeyOnSide2
ListenPort = 51825

# Node 111
[Peer]
PublicKey = ThisIsThePublicKeyFromSide1
AllowedIPs = 0.0.0.0/0
```

Next, setup the VPN interfaces on each side to auto-start as part of the system. The below example will be for *Debian/Ubuntu*, but other Linux distribution are similar.

Side 1 (Node 111):

- Edit the file `/etc/network/interfaces.d/wg222.conf`. Page the content as follows:

```
auto wg222
iface wg222 inet static
        address 10.70.xx.0/31
        pre-up ip link add $IFACE type wireguard
        pre-up wg setconf $IFACE /etc/wireguard/$IFACE.conf
        pre-up ip link set up dev $IFACE
        pre-up ip route add 10.70.xx.1/31 dev $IFACE table nycmesh
        pre-up ip rule add iif $IFACE pref 1031 table nycmesh
        pre-up ip rule add from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
        post-down ip rule del iif $IFACE table nycmesh
        post-down ip rule del from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
```

Side 2 (Node 222):

- Edit the file `/etc/network/interfaces.d/wg111.conf`. Page the content as follows:

```
auto wg111
iface wg111 inet static
        address 10.70.xx.1/31
        pre-up ip link add $IFACE type wireguard
        pre-up wg setconf $IFACE /etc/wireguard/$IFACE.conf
        pre-up ip link set up dev $IFACE
        pre-up ip route add 10.70.xx.1/31 dev $IFACE table nycmesh
        pre-up ip rule add iif $IFACE pref 1031 table nycmesh
        pre-up ip rule add from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
        post-down ip rule del iif $IFACE table nycmesh
        post-down ip rule del from 10.70.xx.0 table nycmesh
        post-down ip link del $IFACE
```

Bring Up the interface on each side like so:

```
#Side1# ifup wg222
#Side2# ifup wg111
```

To verify the tunnel is working, a ping from one side's address to the other side will yield a response. For Example:

```
#Side1# ping -c 2 10.70.89.1
PING 10.70.xx.1 (10.70.xx.1) 56(84) bytes of data.
64 bytes from 10.70.xx.1: icmp_seq=1 ttl=64 time=5.28 ms
64 bytes from 10.70.xx.1: icmp_seq=2 ttl=64 time=5.67 ms

--- 10.70.xx.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 5.280/5.479/5.678/0.199 ms
```

This indicates the tunnel is setup successfully.

Routing Daemon - Bird - OSPF

The routing daemon (BIRD is recommended) needs to be configured to properly convey the routes to/from the network.

Remember, it is important to not leak routes in or out of the network, especially if the VPN node it to be a transit node connecting far sides of the mesh.

Additionally, BIRD's OSPF implementation is a bit finicky -- For example, it does not fully support PTMP mode, unless it communicates with other BIRD instances.

Here is a recommended BIRD configuration for a VPN connection. Key lines will be discussed in-line in the comments.

```
# The router id should be the Node's 10-69-net IP.
# If the node number is 1234:
# If this is the first router, the IP is 10.69.12.34
# If this is the second router, the IP is 10.69.12.134 ( more likely )
# If this is the third router, the IP is 10.69.12.234 ( rare case; Last octet needs to be <255
)
# Important: Use this same number in your local mesh interface above!
router id <Our 10-69-net IP>;

protocol device {
        scan time 10;
}

# Add Local interfaces here which will connect to other local routers, such as an Omni
protocol direct {
```

```
        interface "<Local Interface>";
        interface "wg*";
        interface "dummy*";
}

protocol kernel {
        # Only add this line below if using the multiple routing table method.
        # Note this is referenced by number, use the correct number table from rt_tables file
        kernel table 2;
        scan time 10;
        persist;
        learn;
        metric 64;
        import none;
        export filter {
                if source = RTS_STATIC then reject;
                accept;
        };
}

protocol ospf {
        import all;
        export none;

        area 0 {
                # Add this interface clause for each local interface connecting to local
routers
                interface "<Local Interface>" {
                        # Cost of 1 is safe for this because it's just a local jumper to
another router which has cost
                        cost 1;
                        # Use PtP is going to a Mikrotik Router. BIRD and Mikrotik dont speak
the same PTMP
                        # Use PtMP if going to other BIRD instances
                        # Use Broadcast for special scenarios that make sense, such as at a
supernode.
                        type ptp;
                        # Add Neighbors via IPs on that interface.
                        neighbors {
```

```
                                <Local Neighbor>;
                        };
                };
                # Make sure to use the "wgXXXX" interface for the remove node.
                # Add one of these clauses for each wireguard connection
                interface "wg<Other Node Number>" {
                        # Cost 15 is for a Really Good WireGuard connection. Cost of 40 is
more typical for a VPN
                        cost 15;
                        # PTMP for other BIRD instances. If WireGuard it's gonna be linux so
BIRD
                        type ptmp;
                        neighbors {
                                <Other WireGuard Node>;
                        };
                };
        };
}
```

Testing the configuration

Once everything is running, the WireGuard tunnel is operating, and BIRD is started, check to see if everything is connecting properly. It may be a good idea to *not* initially connect other mesh routers to the VPN router until after you verify it is communicating properly and not leaking routes.

Check BIRD functionality.

Start BIRD and check the OSPF protocol, see below:

```
# birdc
bird> show protocol
name     proto    table    state  since       info
device1  Device   master   up     2020-03-14
direct1  Direct   master   up     2020-03-14
static1  Static   master   up     2020-03-14
kernel1  Kernel   master   up     2020-03-14
ospf1    OSPF     master   up     2020-03-14  Running
bird> show protocol all ospf1
```

```
name     proto    table    state  since       info
ospf1    OSPF     master   up     2020-03-14  Running
  Preference:     150
  Input filter:   ACCEPT
  Output filter:  REJECT
  Routes:         464 imported, 0 exported, 461 preferred
  Route change stats:     received   rejected   filtered    ignored   accepted
    Import updates:          64065          1          0          0      64064
    Import withdraws:        16846          0        ---          2      16845
    Export updates:          64067      64059          8        ---          0
    Export withdraws:        16845        ---        ---        ---          0
bird> show route export ospf1
bird>
```

Above, we see `show protocol` lists `ospf1` as `Running`. This means is has successfully connected to other nodes and is functioning properly. It may also say `alone` which could indicate a problem, or, could mean it has not yet connected (it maybe take up to one minute).

Next we see that inspecting the ospf protocol more, we have received 464 routes, and **are exporting 0 routes** This is the important part. In thie case exporting means we are adding some routes to the mesh. We may want to, for special cases, at this point, but if the VPN router is pass-through, then it should say **0** as a best practice.

Lastly, if we are exporting routes, we can check them with `show route export ospf1`. In this case we aren't so there's no problem. If we do, we can fix the problem with this knowledge.

There are many other commands within BIRD to help debug, check them out.

Revision #3
Created 9 December 2023 04:39:53 by Willard Nilges
Updated 11 June 2024 04:08:42 by James