wg_tool

Release 8.0.0

Gene C

Sep 17, 2025

CONTENTS

CHAPTER

ONE

WG-TOOL

1.1 Overview

Wireguard is a superb vpn built in to the linux kernel. It is robust and super fast. While the majority of wireguard servers run on linux, they are found on other operating systems as well. Most platforms, from desktops to mobile phones, have wireguard clients available. We found wireguard to be more robust and faster than many legacy types of vpns.

A wireguard network is a collection of *peers* that are able to securely communicate with one another. Some peers, known as gateways, allow other peers to connect to them. Peers that are not gateways are called clients.

A basic vpn has a wireguard gateway and several clients that can use the gateway. The gateway may provide access to internel networks. It may also allow clients to have all their internet traffic flow via the gateway.

Since traffic between the client and the gateway is encrypted, this provides privacy and security. Wireguard crypto mechanisms include pre-shared keys (*PSKs*) that provide additional resistance against post quantum attacks. For maximum security a *PSK* should be unique to each pair of peers that communicate with one another.

wg-tool is a wireguard configuration tool that simplifies managing all aspects of wireguard vpn setups.

It handles multiple gateways (e.g. office locations), sharing of internal networks as well as split tunneling (internet via vpn or not).

It guarantees that all gateway and client configs remain consistent with one another. Two peers that communicate with one another use encryption and therefore they share key information. By consistent we mean that both of those peers always use the same public and pre-shared keys.

While *PSKs* are not mandatory in wireguard, they do enhance security. There can be many pairs of *peers* with each pair sharing a unique *PSK*. *wg-tool* quietly takes care of all of them for you.

Wireguard gateway reports (created by running *wg show*) identify peers by their public key. The tool can display these reports using their corresponding user friendly names. Solving a long standing, if minor, *annoyance*.

We built *wg-tool* to make our own VPN administration simple and robust.

1.1.1 Where to Get wg-tool

Available at:

- wg_tool
- Archlinux wg_tool

On Archlinux it can be installed from the AUR or using the PKGBUILD provided in packaging directory.

All git tags are signed with an arch@sapience.com key available via WKD or from the sapience.com website. Add the key to your package builder gpg keyring. The key is included in the Arch package and the source= line with *?signed* at the end can be used to verify the git tag. You can also manually verify the signature as usual with *git tag -v <tag>*.

For those with linux road warriers, there is a wg-client companion package. This is a linux client command line tool packaged with a graphical program that makes it very simple to start and stop a wireguard client for any user.

We offer three working examples. In each example the goals are explained followed by a walk through using *wg-tool*. The resulting standard wireguard configs are then provided .

See *Working Examples* section.

Documentation source along with pre-generated PDF and html versions are in the *Docs* directory. All documentation is written using restructured text.

1.2 Key features

- Simplifies wireguard administration.
- Guarantees gateway and peer configs remain synchronized (public/preshared keys).
- Handles key generation and updates.
- Each *account* can have multiple profiles (vpn1.alice.laptop vpn1.alice.phone etc.)
- Accounts and/or profiles can be marked (in)active.
- Wireguard's standard 'wg show'report has public keys transformed into user friendly acccount.profile names.

  This feature solves a long standing wireguard annoyance in a simple way by showing names instead of public keys output by *wg show*. Also provides check on server config status being current and if it needs to be restarted with new config.
- Supports importing from existing wireguard config files.

1.3 New / Interesting

Major Version 8.0.0

- Re-write pretty much from scratch. New design and fresh start.
- Modern coding standards: PEP-8, PEP-257 and PEP-484 style and type annotations
- Can now manage multiple VPN's
- Each VPN has a number of *accounts* and each account may have multiple profiles. Some profiles may be gateways (can be connected to) while others are clients (connect to one or more gateways).
- Support more use cases than earlier versions.
- Provide walk through *Working Examples* for 3 common use cases
- The enhancements require significant data format changes.

  To make the upgrade as simple and easy as possble, existing data from earlier versions can be auto migrated to the new format with the *-migrate* option.

  Please see *Migrating from Earlier Version* for more details.
- Network manipulations are now built on the *py-cidr* module. Available at py-cidr and Archlinux py-cidr AUR.
- New way to modify profiles.

  The *–edit* option creates a text file. The file uses standard TOML (key = value) format. Simply edit the file and then use the *–merge* option to incorporate those changes.

  This is simple and clean and makes it easy to modify whatever may be needed in one quick edit and merge.

There are still many command line options which can be particularly helpful making bulk changes.

- Improved command line help.

  Command line option help is now organized by category:

```
migrate, edit/merge, reporting, general and stored options.
```

See *wg-tool –help* for more info or *Command Line Options*.

- Document most features including migration, importing, and making modifications.

CHAPTER

TWO

DOCUMENTATION

2.1 PDF and HTML

The complete documentation is available in *Docs/wg_tool.pdf* as well an html version - just point a browser at *Docs/html*.

The document source is also available to build your own:

```
make latexpdf; make latexpdf
make html
```

This requires some sphinx packages being available (see *Install*)

CHAPTER

THREE

GETTING STARTED

3.1 Brief Wireguard Background

It may be useful to review the wireguard documentation https://www.wireguard.com/. The key section on *Cryptokey Routing* describes how peers communicate securely with one another. In addition, the man pages for *wg-quick* and *wg* offer a lot of useful information.

Here, we simply highlight a few relevant items to set the stage.

Every entity in wireguard is a *peer*. And *peers* interact with each other. The way they interact is determined by the config files.

Some peers allow others to connect to them at a known hostname or IP address and on a specific port. The *endpoint* for this connection is given by *host:port*. Any such peer that listens on a known *endpoint* is called a **gateway**.

Other peers only connect to peer gateways and we refer to these as **clients**.

Of course gateways may connect with other gateways too. They are still gateways. So, a gateway is simply a peer that listens on some IP address and port.

3.1.1 Wireguard Config: Interface Section

This section is about configs used by wireguard itself.

Each wireguard peer config consists of 2 kinds of components.

The first part of any peer config is the *Interface* section. This specifes the crypto keys and one or more IP addresses used by the vpn tunnel. It may also have a list of DNS servers.

It also has *PostUp* and *PostDown*. Together, these provide a way to have a program run when the vpn is brought up and some other program run when the vpn is taken down.

These are typically used to set up firewall rules (nftables) and are used primarily for gateway servers and linux clients. We provide more information on these later in the documentation.

We also provide a sample nftables firewall script which will suffice for many/most typical gateway servers. We also provide a linux client program which is a DNS helper tool, changing DNS resolution while the vpn is running and restoring it to it's original state when the vpn is stopped.

3.1.2 Wireguard Config: Peer Section(s)

The second part of any peer config is one or more *Peer* sections. Each of these provides the information required to engage with that peer. Each peer section has the public key of of the peer.

It also provides the list of networks that are acceptable to use in communicating with that peer. The available networks are typically internal LANs or internet access. These is the *AllowedIPs* variable. It can be one network, a comma separated list of networks, and it can be repeated.

Each pair of peers may also share a secret known as a *pre-shared-key* or PSK. Wireguard's author, Jason Donenfeld, opines that using this provides an additional security layer facilitating post-quantum resistance. *wg-tool* automatically generates a unique PSK for each pair of peers that communicate with one another.

The peer section also includes the Endpoint, if that peer is a gateway.

A client may send all it's traffic to the the gateway it is using or it may choose to send only the internal LAN traffic to the gateway and direcly connect to the internet for the remaining traffic. When the client separates the packets like this, it is known as split tunnelling or split routing.

3.1.3 A Note About Shared Networks

Wireguard denotes networks available to each peer using *AllowedIPs*. That variable tells wireguard to permit packets to from the those networks.

Wireguard uses that to create appropriate routes which all work. This means that if a gateway, for example, offers LAN access to it's clients then every client has its *AllowedIPs* with that LAN network listed.

wg-tool generates those *AllowedIPs* that wireguard needs. We designed it to minimize user input and to keep those inputs aligned with physical reality.

If some gateway offers access to an internal network, there should be no need to edit every peer to add that network. Instead, the tool updates all peers with access to that network.

For example, if the Office A gateway offers LAN-A access to clients, then *wg-tool* simply has the gateway designate LAN-A available to other peers. When it generates the wireguard configs, each peer that asks for access to that network will have access permitted via it's *AllowedIPs*.

wg-tool expects each peer to list those networks it offers to other peers using the *nets_offerered* variable. Of course, this is entirely optional, and only needs to be used by peers that wish to share one or more networks. This applies to both gateways and clients.

In addition, gateways may set the *internet_offered* flag to indicate that it will pass traffic to and from the internet on behalf of it's clients. Clients, in turn, request such access using the *internet_wanted* flag.

Similarly, peers may request access to one or more networks by setting *nets_wanted*.

Based on all that information, *wg-tool* generates the appropriate *AllowedIPs* in the wireguard configs.

3.1.4 Peer to Peer

By default peers are only permitted to communcicate with gateways. If it is desirable to allow peers to communicate with one another then this is easily achieved. Please see the *peer_to_peer* vpn info variable in the *Editing and Making Changes* section for more detail.

3.2 Migrating from earlier versions

Versions prior to the **8.0** can be migrated to the new format with *wg-tool –migrate*.

The migration leaves all earlier data files, including the output wireguard configs, untouched. The current version uses different directories (*Data* and *Date-wg*) to guarantee this.

3.3 Importing from standard wireguard configs

For those with existing wireguard setup, *wg-tool* can import standard wireguard configs. Once imported then *wg-tool* can be used to manage things going forward.

For more info please see *Importing From Standard Wireguard Configs*.

After all the configs are imported, its helpful to compare the resulting generated configs (in *Data-wg*) with those that were imported.

3.4 Simple Example

Lets do a really little example that illustrates how easy it is to generate wireguard configs. The goal here is to:

- Create a vpn called *vpn-test*
- Add an account called servers with a gateway called *wg-A*
- Add an account called *alice* with a laptop profile.
- Add an account called *bob* with a laptop profile.
- Do this in the current directory.

```
wg-tool -wkd ./
wg-tool -new vpn-test
wg-tool -new vpn-test.servers.wg-A
wg-tool -new vpn-test.alice.laptop vpn-test.bob.laptop
```

Add the endpoint the gateway server will be available on:

```
wg-too --edit vpn-test.servers.wg-A
```

Edit the file (name will be displayed) and change the Endpoint to something like:

```
Endpoint = "vpn.example.com:51820"
```

Then merge the change:

```
wg-tool --merge <filename>
```

All the wireguard configs will be found under the *Data-wg* directory. This has the gateway server config along with both users' laptop configs.

CHAPTER

FOUR

LICENSE

wg_tool software is used to administer Wireguard VPNs.

CHAPTER

FIVE

WORKING EXAMPLES

Working examples often provide a good learning path. Here we offer three such examaples, each offering a step by step illustration how to achieve each goal.

In each case, we lay out the goal and then walk through using *wg-tool* to achieve that goal.

At the end the generated standard wireguard configs are shown.

The first example is the simplest with a single gateway, A, and some road warrier clients. The gateway provides it's clients with access to Office A LAN and an option for their internet traffic to go via the gateway.

The second example extends this adding a second gateway, B. This new gateway provides access to a second Office LAN. Clients now access Offie A's LAN via gateway A and Office B's LAN via gateway B.

The last example modifies gateway B to be a client. This client connects to gateway A and still provides Office B LAN access but now only to gateway A which in turn offers access to it's own clients.

Gateway A is now able to provide access for road warrier clients to both LAN A and LAN B with help from *client B*.

I would encourage you to work through the examples. But, if you want to quickly see the results of doing the examples, there are scripts in the examples directory that do it for you. The scripts generate the data in the same directory they are run in.

```
create-example-1
create-example-2
create-example-3
```

These can be run sequentially starting with *create-example-1*. You can then run *create-example-2* and *create-example-3*. After each step you may want to look at the outputs saved under *Data-wg* and get a summary using:

```
wg-tool --list
```

5.1 Where to Run The Gateway

While gateways can be run on a firewall / border router, we prefer to run them behind the firewall, where the firewall forwards necesary UDP port.

The reason we like this approach is that firewall rules used by the gateway don't have to be concerned about any existing firewall rules, as is the case when gateway runs on the firewall itself. It's also a little cleaner from a security perspective to minimize what runs on the firewall.

One downside putting the vpn behind the firewall is that internal hosts wishing to use the gateway, may need additional routes.

For example, if LAN-A clients wish to access LAN-B, they will need a route to the appropriate vpn gateway to allow this.

In contrast, when the vpn runs on the firewall, since the firewall is typically the default route, additional routes may not be needed.

Adding static routes in linux is straightforward.

For DHCP, ISC's Kea supports classless static routing[3]. In Kea this takes a list of routes, each of the form:

```
network - IP-address
```

which signifies that the route to *network* is provided by *IP-address*. Per the RFC this option supercedes the older routers option and thus all routes should be listed in addition to having the older option.

If the gateway runs on the firewall, and LAN clients have a default route to the firewall, then it all works without any additional route.

For the working examples, exactly where the gateways run plays no role at all.

On to the examples!

5.2 Working Example 1

This example has a single wireguard gateway, *wg-A*, providing access to Office-A internal LAN, *192.168.1.0/24* to it's clients. It also offers internet access for any clients wanting all internet traffic to go through the vpn.

This expands on the rather *Simple Example* shown earlier.

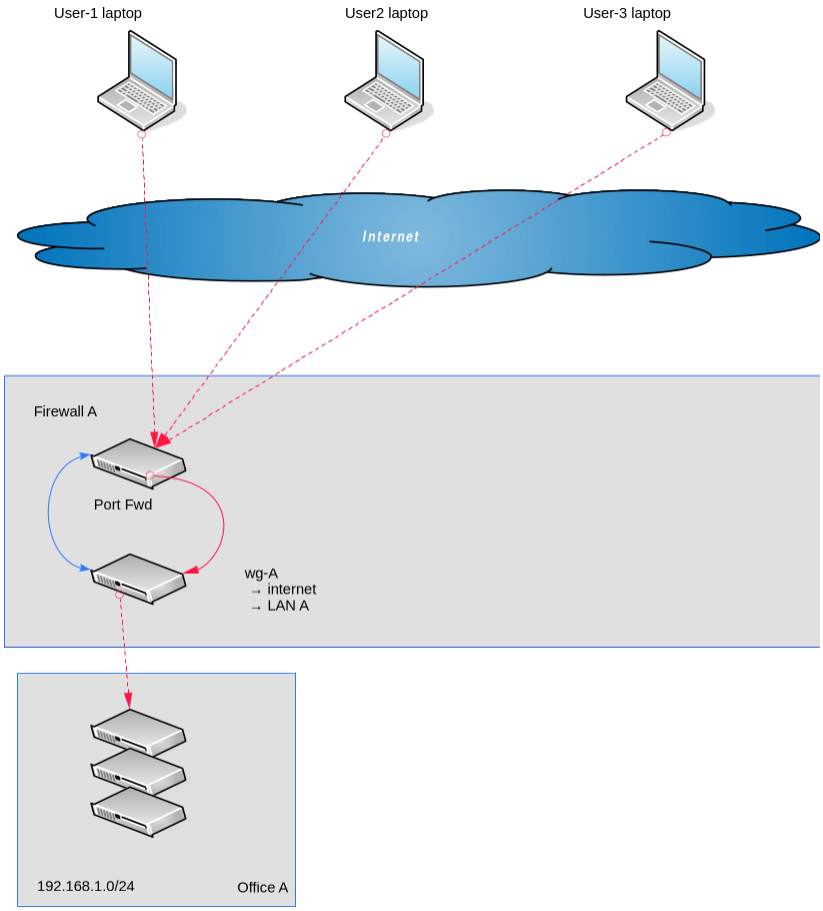
As we do in all the examples, we first walk through *Generating the Wireguard Configs Using wg-tool*.

At the end of the walk throgh, we show the resulting *Standard Wireguard Configs* that are created. In this example the result will be the gateway and it's client configs. Client configs come with a QR code carrying the same information.

These configs are the ones consumed by wireguard itself.

[3] https://datatracker.ietf.org/doc/html/rfc3442

5.2.1 Network Diagram



We find it helpful to make a small table with the goals for this vpn.

The table lists each of the peers and what they want from or offer to other peers.

Gateways are those peers which other connect to. All gateways are marked. Some gateways may provide internet access.

Some peers may offer network access to specific networks.

Some clients may want their internet traffic to flow through the vpn while others may want to access the internet directly and use the vpn soley to access internal networks.

For all examples we use a vpn named *vpn-test*.

5.2.2 Goals Table

Table 1: Example 1 Goals

Account	Profile	Gateway	Internet wanted	offered	Networks offered	wanted
servers	wg-A	✓	✗	✓	192.168.1.0/24	-
user-1	laptop	✗	✓	✗	-	192.168.1.0/24
user-2	laptop	✗	✓	✗	-	192.168.1.0/24
user-3	laptop	✗	✗	✗	-	192.168.1.32/28

wg-A also offers internet access to it's peers. Some peers, such as *user-3.laptop* may prefer to access the internet directly and only use *wg-A* to access *LAN-A*.

user-3.laptop is using *split routing* (also called *split tunnelling*), since it routes *LAN-A* traffic over the VPN, while all other traffic goes direct.

user-1.laptop, on the other hand has all of it's network traffic go via *wg-A*.

In this example we use the following networks:

- *vpn.example.com:51820* : Endpoint of gateway *wg-A*
- *10.77.77.0/24* : vpn internal network.
- *192.168.1.0/24* : *LAN-A* the internal Office A LAN.

5.2.3 Generating the Wireguard Configs Using wg-tool

This is a step by step walk through using the tool to create the configs that wireguard uses. Once done you can browse the configs that result.

Please never edit any files in Data/. To make a change, or to view the editable variables, use *wg-tool –edit <ID>*.

That saves a file in the Edits/ directory which you can edit and, if you so choose, merge any changes back with *wg-tool –merge <file>*.

To create your own example, work in an empty directory of your choosing. First create a "vpn" group hich we'll label *vpn-test* here. This contains all related wireguard peers. A peer can be a gateway or a client.

Each peer is is denoted by an identifier which takes the form:

```
<vpn name>.<account name>.<profile name>
```

Names are alphanumeric, with some restrictions on special characters[1]. For example periods are not allowed (for obvious reasons).

We find it convenient to use *servers* for the *account name* for things that run on a server, like gateways. User names make a good account name choice for people.

There are some instances where an ID may be shortened. For example to create a new vpn, we can drop account and/or profile part. Later we use the full ID to create gateway and client profiles.

[1] Valid names comprise letters, numbers and = - _ ~ + ; : @

Make Working Directory

The first step is to establish a working directory:

```
mkdir my-examples
cd my-examples
wg-tool -wkd ./
```

Note that networks in the *AllowedIPs* wireguard config are compacted to their minimal CIDR representation. The pre-compacted list, perhaps most useful during testing, is provded as a comment in the config file. For additional information please see *Compacting Networks*.

Make New VPN called vpn-test

We now create a new vpn named *vpn-test*. For this the ID is simply *vpn-test*:

```
wg-tool -new vpn-test
```

This also invokes the *edit* option and displays the file name. This file will be ./Edits/vpn-test-info.mods.

Please edit this file and add any relevant VPN information. In our case we really may only need to add a DNS server (or 2), host or IP.

```
dns = ['10.10.10.10']
peer_to_peer = false
```

Note that elements in *dns* are dns servers. They can be an IP address or a hostname; hostnames will be converted to their IP address using local dns resolver.

For more information about DNS please see *Note On DNS*.

You can change the internal tunnel network(s) if you prefer or leave the default setting.

Leave the peer_to_peer set to *false* unless you want peers to communicate directly with one another, rather than just with the gateway(s).

The effect of turning on peer to peer is to change the wireguard *AllowedIPs* variable to permit the entire vpn network (*/24 or */64* for IPv6) instead of a single IP address for every peer.

Leave it to *false* to match the example configs at the given at the end. This is the typical setting.

Never modify the *tag* field - it is a unique identifier used to ensure any edits go to the correct place.

Next merge any changes:

```
wg-tool --merge Edits/vpn-test-info.mods
```

You can make changes at any time using the *–edit* option which saves a then current version to the file.

You can change this at any time using *wg-tool –edit vpn-test*, modify the setting and *-merge* the changes back.

Add Gatewway wg-A

Next we add the gateway and some clients. The tool handles all keys and internal vpn IP addresses.

We'll use an *account* called *servers* with one gateway profile, *wg-A*.

So now do:

```
wg-tool --new vpn-test.servers.wg-A
wg-tool --edit vpn-test.servers.wg-A
```

Edit the file, filename is displayed. In this case it will be *Edits/vpn-test.servers.wg-A.mods*.

Since every gateway must be reachable, an *Endpoint* is required. We also want to provide access to Office A's internal LAN.

wg-A has access to an internal network which we will allow it to share with other peers. We do this by setting the *internet_offered* with the network. This can be a list of networks, but we are only providing access to one.

This means we need to modify a few lines as shown:

```
Endpoint = "vpn_A.example.com:51820"
internet_offered = true
internet_wanted = false
nets_offered = ["192.168.1.0/24"]
post_up = ['/usr/bin/nft -f /etc/wireguard/scripts/postup.nft']
post_down = ['/usr/bin/nft flush ruleset']
```

The *post_up/down* are scripts wireguard runs bringing vpn tunnel up/down. The nftables rules allow traffic to be NAT'd to and from the tunnel. If you wish, you can add another DNS server here as well (dns=..).

As we did with vpn edits, merge these changes back using:

```
wg-tool --merge Edits/vpn-test.servers.wg-A.mods
```

At this point there will be a wireguard config for *wg-A* in

```
Data-wg/vpn-test/servers/wg-A.conf
```

It will have an *Interface* section but no peers, since we haven't yet created anything else beside the one gateway.

Add 3 Clients Peers

Now we're ready to add some client profiles.

We create 3 users where eech user has one profile named *laptop*. We choose *user-1*, *user-2* and *user-3* for the 3 account names.

```
wg-tool -new vpn-test.user-1.laptop vpn-test.user-2.laptop vpn-test.user-3.laptop
```

Making LAN Network Available to Clients

Split Routing

Lets also make user-3.latpop use split routing

By default the laptop clients route all their traffic via the gateway - both for LAN as well as internet traffic.

Lets edit *vpn-test.client-3:laptop* to change it to split rouing.. This means LAN traffic will be routed via the vpn and all regular internet traffic is routed directly.

As usual edit the ID:

```
wg-tool --edit vpn-test.user-3.laptop
```

and modify the file ./Edits/vpn-test.user-3.laptop.mods:

```
internet_wanted = false
```

Then merge the change back. The *edit* option always prints the filename to be edited and merged:

```
wg-tool --merge Edits/vpn-test.user-3.laptop.mods
```

Network Access

wg-A is offering to share its' LAN, *192.168.1.0/24*. We now confgure each of the user laptop configs to request access to it.

We could use *–edit* for each of the IDs to accomplish this. Here we would set the variable *nets_wanted = ["192.168.1.0/24"]* for each and then use *–merge* again.

Alternatively, we can use a command line option, which for this change is a little easier to do.

Lets use the command line option to assign network access for each of the clients.

We will permission user-1.laptop and user-2.laptop with access to *192.168.1.0/24*.

And, for fun, we limit user-3.laptop to the subnet *19.168.1.32/28*

```
wg-tool --nets-wanted-add "192.168.1.0/24" vpn-test.user-1.laptop vpn-test.user-2.
↪laptop
wg-tool --nets-wanted-add "192.168.1.32/28" vpn-test.user-3.laptop
```

We're done.

Listing What We Have

We can show everything looks good by using *–list* (or *-l*) option.

You can add *-v* or *-vv* to get more verbose output. Using *-lv* will also show networks while *-lvv* will show public keys and any *hidden* items. You can also filter what is shown by <vpn>.<account> to show all profiles or <vpn>.<account>.<profile>

For example,

```
wg-tool -lv
```

And the output should look similar to this:

```
Loading vpn-test
 ✓ vpn-test          250914-202654:

    ✓ servers        250914-202654:
                      ✓ wg-A                 250914-202702 (gateway)
                              nets_offered : ['192.168.1.0/24', 'internet']

    ✓ user-1         250914-202655:
                      ✓ laptop               250914-202702
                              nets_wanted : ['192.168.1.0/24', 'internet']

    ✓ user-2         250914-202655:
                      ✓ laptop               250914-202702
                              nets_wanted : ['192.168.1.0/24', 'internet']

    ✓ user-3         250914-202655:
                      ✓ laptop               250914-202702
                              nets_wanted : ['192.168.1.32/28']
```

The date/times are the last modification time. The terminal output may show some ascii colors.

All being well, this will show 1 vpn (*vpn-test*) with 1 gateway under servers.wg-A. It should also show 3 users each with a laptop profile.

Wireguard Config Files

The standard wireguard config files are all saved under the *Data-wg* directory.

In our case the configs files are under *Data-wg/vpn-test/*. The config for the *wg-A* gateway will be in:

```
Data-wg/vpn-test/servers/wg-A.conf
```

while client-1 will be under

```
Data-wg/vpn-test/user-1/laptop.conf
```

and similarly for the other 2 clients.

All being well, these wireguard config files will match those we showed at in the next section.

Configs as QR Code

For non-gatways, a QR code version of the same config is generated under each account's *qr* directory. For example:

```
cat Data-wg/vpn-test/user-1/laptop.conf
Data-wg/vpn-test/user-1/qr/laptop.png
```

You will also see that every pair of peers has it's own unique pre-shared key. This is used for all communications between those peers and adds additional security against post-quantum attacks.

The PSK shared between a gateway and a client will be in the gateway config in the *Peer* section for the client and the same key will be in the client config in the *Peer* section for that gateway.

These all have both IPv4 as well as IPv6 tunnel addresses automatically generated. We skipped when listing the configs at the begining of this example.

Wireguard client apps usually can take either the text *.conf* file or use a camera to read the QR code. The QR code has exactly the same information as the corresponding *.conf* file.

You can view the QR image and see the content of a QR code using, for example[2]:

```
zbarimg --raw Data-wg/vpn-test/user-1/qr/laptop.png
```

This displays the same data as the config, without any comments which we stripped out before building the QR code.

5.2.4 Standard Wireguard Configs

Taken from *Data-wg/*....

Note that leading whitespace should not be used in actual config files. In this document we added extra white space to the output configs solely to aid in visually separating the sections from one another.

wg-A

- cat Data-wg/vpn-test/servers/wg-A.conf

[2] zbarimg provided by the *zbar* package.

```
[Interface]          # servers wg-A (gateway)
    PrivateKey           = <privkey>
    ListenPort           = 51820
    Address              = 10.77.77.1/24, fc00:77:77::1/64
    PostUp               = /usr/bin/nft -f /etc/wireguard/scripts/postup.nft
    PostDown             = /usr/bin/nft flush ruleset


#
# Clients
#

[Peer]               # user-1 laptop
    PublicKey            = <pubkey user-1.laptop>
    PresharedKey         = <psk wg-A x user-1.laptop>
    AllowedIPs           = 10.77.77.2/32, 192.168.1.0/24, fc00:77:77::2/128

[Peer]               # user-2 laptop
    PublicKey            = <pubkey user-2.laptop>
    PresharedKey         = <psk wg-A x user-2.laptop>
    AllowedIPs           = 10.77.77.3/32, 192.168.1.0/24, fc00:77:77::3/128

[Peer]               # user-3 laptop
    PublicKey            = <pubkey user-3.laptop>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.4/32, 192.168.1.32/28, fc00:77:77::4/128
```

user-1

- cat Data-wg/vpn-test/user-1/laptop.conf

```
[Interface]          # user-1 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.2/32, fc00:77:77::2/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-1.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
    # pre-compacted        ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-2

- cat Data-wg/vpn-test/user-2/laptop.conf

```
[Interface]          # user-2 laptop
    PrivateKey           = <privkey>
```

(continues on next page)

(continued from previous page)

```
    Address              = 10.77.77.3/32, fc00:77:77::3/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-2.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
    # pre-compacted        ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-3

- cat Data-wg/vpn-test/user-3/laptop.conf
- using split routing

```
[Interface]          # user-3 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.4/32, fc00:77:77::4/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.1/32, 192.168.1.32/28, fc00:77:77::1/128
    Endpoint             = vpn_A.example.com:51820
```

5.3 Working Example 2

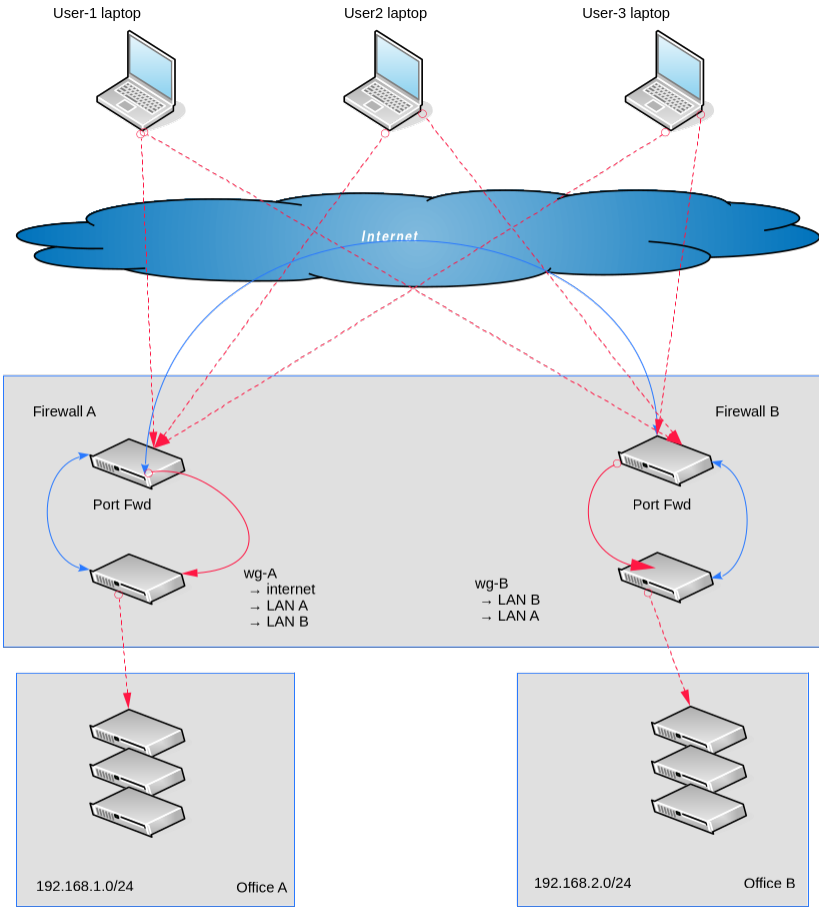
This example builds on *Working Example 1* by adding second gateway.

The new gateway *wg-B* provides access to Office-B LAN, *192.168.2.0/24*. Both gateways provide internet access and are visible on the internet.

wg-A still offers access to Office A LAN.

We walk through the steps *Generating the Wireguard Configs Using wg-tool* which leads to the creation of the *Standard Wireguard Configs*.

5.3.1 Network Diagram



user-1 and user-2's laptops can connect to both gateways and access both LANs. Their internet traffic will go through the vpn, providing privacy. Their internet access could be one or the other gateway.

Note that wireguard relies on the networks shown in the config files to set up routing. It does not check nor provide a failover mechanism or any dynamic checks and adjustments.

In practice wireguard chooses the first peer offering a route to the internet and stick with that. This happens when wireguard starts.

user-3 continues to use split routing, retaining direct internet access and using the gateways solely for LAN access to both office locations.

As in the previous example, we construct the goals table:

5.3.2 Goals Table

Table 2: Example 2 Goals

Account	Profile	Gateway	Internet wanted	offered	Networks offered	wanted
servers	wg-A	✓	✗	✓	192.168.1.0/24	192.168.2.0/24
servers	wg-B	✓	✗	✓	192.168.2.0/24	192.168.1.0/24
user-1	laptop	✗	✓	✗	-	192.168.1.0/24
						192.168.2.0/24
user-2	laptop	✗	✓	✗	-	192.168.1.0/24
						192.168.2.0/24
user-3	laptop	✗	✗	✗	-	192.168.1.32/28
						192.168.2.0/24

5.3.3 Generating the Wireguard Configs Using wg-tool

Next we create the wireguard configs above using wg-tool. This will be quit a bit simpler than editing all those configs manually.

We build on what was done in Part 1 working in the same directory.

Add Second Gateway wg-B

Lets add the new gateway *wg-B*

```
wg-tool --new vpn-test.servers.wg-B
wg-tool --edit vpn-test.servers.wg-B
```

Edit the file as displayed, this time it will be *Edits/vpn-test.servers.wg-B.mods*.

As we did for *wg-A*, add an Endpoint for *wg-B*.

```
Endpoint = "vpn_B.example.net:51820"]
internet_offered = true
post_up = ['/usr/bin/nft -f /etc/wireguard/scripts/postup.nft']
post_down = ['/usr/bin/nft flush ruleset']
```

As before merge the changes back:

```
wg-tool --merge Edits/vpn-test.servers.wg-B.mods
```

Network Access

First lets set *wg-B* to offer Ofice B LAN network, *192.168.2.0/24*, and also give it access to the LAN offered by *wg-A* network.

We'll also give *wg-A* access to this network, *192.168.2.0/24*.

I other words, each of the 2 gateways will have access to the LAN provided by the other gateway.

Lets use the command line options to accomplish this.

```
wg-tool --nets-offered-add "192.168.2.0/24" vpn-test.servers.wg-B
wg-tool --nets-wanted-add "192.168.1.0/24" vpn-test.servers.wg-B
```

(continues on next page)

(continued from previous page)

```
wg-tool --nets-wanted-add "192.168.2.0/24" vpn-test.servers.wg-A
```

Next we add permission to *192.168.2.0/24* for all 3 user profiles.

```
wg-tool --nets-wanted-add "192.168.2.0/24" vpn-test.user-1 vpn-test.user-2 vpn-test.
↪user-3
```

Notice that we used *<vpn>.<account>* for the user IDs. This is a shortcut for every profile belonging to *<vpn>.<account>*. In our case each user only has a single profile, laptop. If there were more then, every profile would have this network added.

You can add multiple networks using *–nets-wanted-add* by separating them with commas. For example you can use "102.168.2.0/24,192.168.100.0/24" to add 2 networks.

Listing What We Have

As we did in previous example, lets list everthing to confirm it has what we want.

```
wg-tool -lv
```

Output should look similar to this:

```
✓ vpn-test          250915-065904:

    ✓ servers       250915-093519:
                      ✓ wg-A                250915-093526 (gateway)
                                          nets_offered : ['192.168.1.0/24', 'internet']
                                           nets_wanted : ['192.168.2.0/24']
                      ✓ wg-B                250915-093526 (gateway)
                                          nets_offered : ['192.168.2.0/24', 'internet']
                                           nets_wanted : ['192.168.1.0/24']

    ✓ user-1        250915-065905:
                      ✓ laptop              250915-093526
                                           nets_wanted : ['192.168.1.0/24', '192.168.2.0/24
↪', 'internet']

    ✓ user-2        250915-065905:
                      ✓ laptop              250915-093526
                                           nets_wanted : ['192.168.1.0/24', '192.168.2.0/24
↪', 'internet']

    ✓ user-3        250915-065905:
                      ✓ laptop              250915-093526
                                           nets_wanted : ['192.168.1.32/28', '192.168.2.0/
↪24']
```

This shows both gateways sharing one another's networks and the users have also gained access to *192.168.2.0/24*

vpn-test.user-3.laptop is still using split tunnelling.

As before, the resulting wireguard config files reside under *Data-wg/vpn-test1*.

By now you may feel the list output above is sufficient to confirm things are as intended, but for completeness we'll show the resulting wireguard configs, as we did in the previous example.

The config for the 2 gateways *wg-A* and *wg-B* will be in:

```
cat Data-wg/vpn-test/servers/wg-A.conf
cat Data-wg/vpn-test/servers/wg-B.conf
```

while the 3 user acoounts and their laptop profiles will be found:

```
cat Data-wg/vpn-test/user-1/laptop.conf
cat Data-wg/vpn-test/user-2/laptop.conf
cat Data-wg/vpn-test/user-3/laptop.conf
```

These files should match what is shown below.

5.3.4 Standard Wireguard Configs

wg-A

```
[Interface]          # servers wg-A (gateway)
    PrivateKey           = <privkey>
    ListenPort           = 51820
    Address              = 10.77.77.1/24, fc00:77:77::1/64
    PostUp               = /usr/bin/nft -f /etc/wireguard/scripts/postup.nft
    PostDown             = /usr/bin/nft flush ruleset

#
# Gateways
#

[Peer]               # servers wg-B (gateway)
    PublicKey            = <pubkey wg-B>
    PresharedKey         = <psk wg-A x wg-B>
    AllowedIPs           = 10.77.77.5/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::5/128
    Endpoint             = vpn_B.example.com:51820

#
# Clients
#

[Peer]               # user-1 laptop
    PublicKey            = <pubkey user-1.laptop>
    PresharedKey         = <psk wg-A x user-1.laptop>
    AllowedIPs           = 10.77.77.2/32, 192.168.1.0/24, fc00:77:77::2/128

[Peer]               # user-2 laptop
    PublicKey            = <pubkey user-2.laptop>
    PresharedKey         = <psk wg-A x user-2.laptop>
    AllowedIPs           = 10.77.77.3/32, 192.168.1.0/24, fc00:77:77::3/128

[Peer]               # user-3 laptop
    PublicKey            = <pubkey user-3.laptop>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.4/32, 192.168.1.0/24, fc00:77:77::4/128
```

wg-B

```
[Interface]          # servers wg-B (gateway)
    PrivateKey           = <privkey>
    ListenPort           = 51820
    Address              = 10.77.77.5/24, fc00:77:77::5/64
    PostUp               = /usr/bin/nft -f /etc/wireguard/scripts/postup.nft
    PostDown             = /usr/bin/nft flush ruleset

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-B x wg-A>
    AllowedIPs           = 10.77.77.1/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::1/128
    Endpoint             = vpn_A.example.com:51820

#
# Clients
#

[Peer]               # user-1 laptop
    PublicKey            = <pubkey user-1.laptop>
    PresharedKey         = <psk wg-B x user-1.laptop>
    AllowedIPs           = 10.77.77.2/32, 192.168.2.0/24, fc00:77:77::2/128

[Peer]               # user-2 laptop
    PublicKey            = <pubkey user-2.laptop>
    PresharedKey         = <psk wg-B x user-2.laptop>
    AllowedIPs           = 10.77.77.3/32, 192.168.2.0/24, fc00:77:77::3/128

[Peer]               # user-3 laptop
    PublicKey            = <pubkey user-3.laptop>
    PresharedKey         = <psk wg-B x user-3.laptop>
    AllowedIPs           = 10.77.77.4/32, 192.168.2.0/24, fc00:77:77::4/128
```

user-1

```
[Interface]          # user-1 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.2/32, fc00:77:77::2/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-B (gateway)
    PublicKey            = <pubkey wg-B>
    PresharedKey         = <psk wg-B x user-1.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.5/32, 192.168.2.0/24
    # pre-compacted        ::/0, fc00:77:77::5/128
```

(continues on next page)

(continued from previous page)

```
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_B.example.com:51820

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-1.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
    # pre-compacted        ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-2

```
[Interface]          # user-2 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.3/32, fc00:77:77::3/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-B (gateway)
    PublicKey            = <pubkey wg-B>
    PresharedKey         = <psk wg-B x user-2.laptop>
    # re-compacted        0.0.0.0/0, 10.77.77.5/32, 192.168.2.0/24
    # re-compacted        ::/0, fc00:77:77::5/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_B.example.com:51820

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-2.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
    # pre-compacted        ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-3

```
[Interface]          # user-3 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.4/32, fc00:77:77::4/128
    DNS                  = 10.10.10.10

#
# Gateways
#
[Peer]               # servers wg-B (gateway)
    PublicKey            = <pubkey wg-B>
    PresharedKey         = <psk wg-B x user-3.laptop>
    AllowedIPs           = 10.77.77.5/32, 192.168.2.0/24, fc00:77:77::5/128
```

(continues on next page)

(continued from previous page)

```
    Endpoint             = vpn_B.example.com:51820

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.1/32, 192.168.1.0/24, fc00:77:77::1/128
    Endpoint             = vpn_A.example.com:51820
```

5.4 Working Example 3

This example makes one small modification to *Working Example 2*.

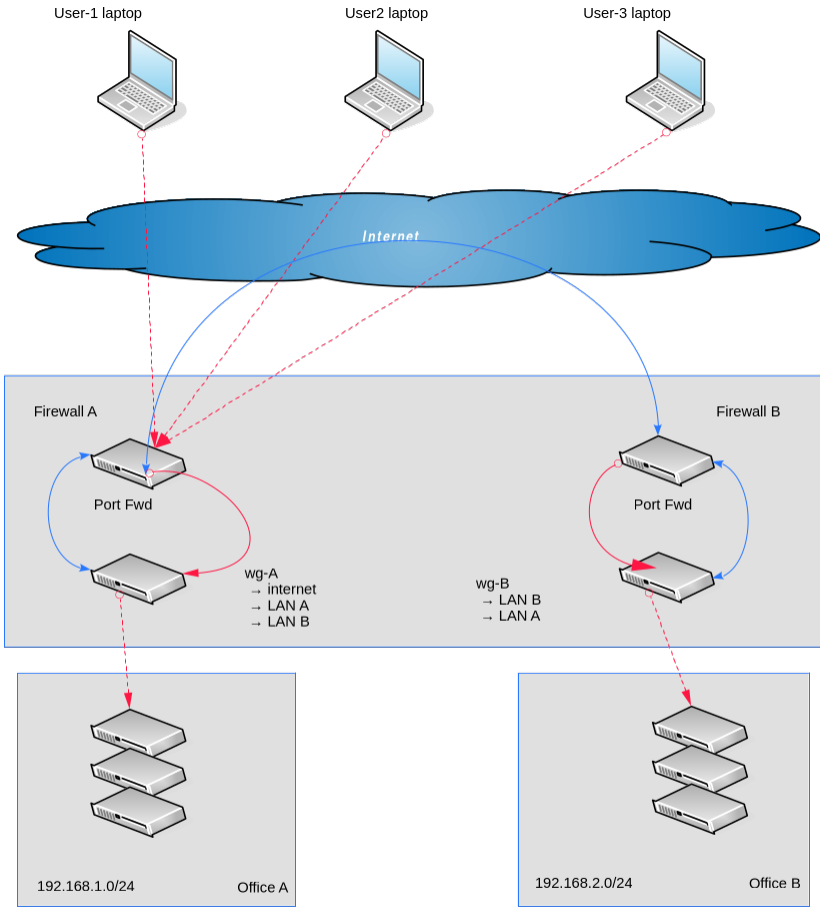
wg-B is changed to be a client instead of a gateway.

This means it no longer has its own *Endpoint*. It continues to offer access to the Office B LAN but now provides it to *wg-A* which in turn will share it with it's clients.

This situation can occur when there are 2 locations, one of them has a public IP address (*wg-A*) while the other one (*wg-B*) does not. Now *wg-B* can connect to *wg-A* as a client, providing access to it's LAN, while *wg-A* offers its LAN to *wg-B*.

The road warrier laptops can only connect to *wg-A* and they all want to have access to both LANs.

5.4.1 Network Diagram



wg-B will be a peer of *wg-A* and offer access to Office B's LAN. However clients will now access this LAN via *wq-A*.

Simple change to the goals, but quite a few changes to the wireguard configs.

When a gateway offers local networks it is straightfoward to make them available to clients. In this case, clients need to be informed that both Office A LAN and Office B LAN are available from gateway *wg-A*.

As before build the goals table:

5.4.2 Goals Table

Table 3: Example 3: vpn-test

Account	Profile	Gateway	Internet wanted	offered	Networks offered	wanted
servers	wg-A	✓	✗	✓	192.168.1.0/24	192.168.2.0/24
					192.168.2.0/24	
servers	wg-B	✗	✗	✗	192.168.2.0/24	192.168.1.0/24
user-1	laptop	✗	✓	✗	-	192.168.1.0/24
						192.168.2.0/24
user-2	laptop	✗	✓	✗	-	192.168.1.0/24
						192.168.2.0/24
user-3	laptop	✗	✗	✗	-	192.168.1.32/28
						192.168.2.0/24

5.4.3 Generating the Wireguard Configs Using wg-tool

As in the previous two examples, we create the wireguard configs using wg-tool. We are going to make a small edit to *wg-B* to achieve this.

```
wg-tool --edit vpn-test.servers.wg-B
```

Edit the file as before, (*Edits/vpn-test.servers.wg-B.mods*). The only change needed is to set *Endpoint* to an empty string and change *internet_offered* to false

```
Endpoint = ""
internet_offered = false
```

Merge the changes:

```
wg-tool --merge Edits/vpn-test.servers.wg-B.mods
```

Since *wg-B* can no longer provide access to other clients, we let *wg-A* do that on it's behalf:

```
wg-tool --nets-offered-add  '192.168.2.0/24' vpn-test.servers.wg-A
```

That should be all that is necessary, *wg-tool* will do the rest.

List what we have:

```
wg-tool -lv
```

The only real difference in the listing is that *wg-B* is no longer showing as a gateway. It isn't of course, its just another peer at this point. However, it still provides access to the *192.168.2.0/24* network.

Here's a terminal screen shot of the output:

```
Loading vpn-test
✓ vpn-test          250915-065904:

    ✓ servers       250915-093519:
                      ✓ wg-A                  250915-201143 (gateway)
                              nets_offered : ['192.168.2.0/24', '192.168.1.0/24', 'internet']
                               nets_wanted : ['192.168.2.0/24']
                      ✓ wg-B                  250915-201143
                              nets_offered : ['192.168.2.0/24']
                               nets_wanted : ['192.168.1.0/24']

    ✓ user-1        250915-065905:
                      ✓ laptop                250915-201143
                               nets_wanted : ['192.168.1.0/24', '192.168.2.0/24', 'internet']

    ✓ user-2        250915-065905:
                      ✓ laptop                250915-201143
                               nets_wanted : ['192.168.1.0/24', '192.168.2.0/24', 'internet']

    ✓ user-3        250915-065905:
                      ✓ laptop                250915-201143
                               nets_wanted : ['192.168.1.32/28', '192.168.2.0/24']
Checking if changes need to be saved
  Checking vpn-test
  Wireguard configs
Cleaning history
Completed
```

As you can see in the verbose list output, *wg-B* is no longer a gateway. It offers access to Office B LAN as a client to *wg-A*. It also *wants* access to Office A LAN.

All laptops want access to both networks which they now get from *wg-A*.

If you look at the wireguard configs you will see that the laptops now get both networks from *wg-A*.

```
cat Data-wg/vpn-test/servers/wg-A.conf
cat Data-wg/vpn-test/servers/wg-B.conf
cat Data-wg/vpn-test/user-1/laptop.conf
cat Data-wg/vpn-test/user-2/laptop.conf
cat Data-wg/vpn-test/user-3/laptop.conf
```

You will see that *wg-B* now has just one peer which is *wg-A*. Furthermore, *wg-A* now allows *192.168.1.0/24* as well as *192.168.2.0/24* to it's clients. Finally, all the laptop clients now have both LANs listed in each *AllowedIPs* with *wg-A*.

Previously, each laptop had LAN A listed for *wg-A* and LAN B for *wg-B*.

If we want to go back with *wg-B* again being a full gateway, then we just edit *wg-B* and restore it's Endpoint and, if desired, allow it to offer internet access as well.

```
Endpoint = "vpn_B.example.net:51820"
internet_offered = true
```

Merge changes back and everything is back to the way it was in *Working Example 2*.

5.4.4 Standard Wireguard Configs

The basic set up is same as example 2. The only thing we're changing, at least superficially, is that *wg-B* is no longer a gateway; it's now another client.

This alsop means other client configs must be changed that they know to access Office B LAN from *wg-A* instead of *wg-B*, as before.

We must also change *wg-B* to remove the Endpoint while still offering LAN-B access.

wg-A

- Note that the Peer section for *wg-B* no longer has an Endpoint

```
[Interface]          # servers wg-A (gateway)
    PrivateKey           = <privkey>
    ListenPort           = 51820
    Address              = 10.77.77.1/24, fc00:77:77::1/64
    PostUp               = /usr/bin/nft -f /etc/wireguard/scripts/postup.nft
    PostDown             = /usr/bin/nft flush ruleset


#
# Clients
#

[Peer]               # servers wg-B
    PublicKey            = <pubkey wg-B>
    PresharedKey         = <psk wg-A x wg-B>
    AllowedIPs           = 10.77.77.5/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::5/128

[Peer]               # user-1 laptop
    PublicKey            = <pubkey user-1.laptop>
    PresharedKey         = <psk wg-A x user-1.laptop>
    AllowedIPs           = 10.77.77.2/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::2/128

[Peer]               # user-2 laptop
    PublicKey            = <pubkey user-2.laptop>
    PresharedKey         = <psk wg-A x user-2.laptop>
    AllowedIPs           = 10.77.77.3/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::3/128

[Peer]               # user-3 laptop
    PublicKey            = <pubkey user-3.laptop>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.4/32, 192.168.1.32/28, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::4/128
```

wg-B

- Client not a gateway
- ListenPort is removed from the Interface section,
- provides access to LAN B
- there are no longer any user laptop sections

```
[Interface]          # servers wg-B
    PrivateKey           = <privkey>
    Address              = 10.77.77.5/32, fc00:77:77::5/128
    DNS                  = 10.10.10.10
    PostUp               = /usr/bin/nft -f /etc/wireguard/scripts/postup.nft
    PostDown             = /usr/bin/nft flush ruleset

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x wg-B>
    AllowedIPs           = 10.77.77.1/32, 192.168.1.0/24, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::1/128
    Endpoint             = vpn_A.example.com:51820
```

user-1

- Note *wg-B* has been removed from all laptop clients and Office B LAN now provided by *wg-A*

```
[Interface]          # user-1 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.2/32, fc00:77:77::2/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-1.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
    # pre-compacted        192.168.2.0/24, ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-2

```
[Interface]          # user-2 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.3/32, fc00:77:77::3/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-2.laptop>
    # pre-compacted        0.0.0.0/0, 10.77.77.1/32, 192.168.1.0/24
```

(continues on next page)

(continued from previous page)

```
    # pre-compacted        192.168.2.0/24, ::/0, fc00:77:77::1/128
    AllowedIPs           = 0.0.0.0/0, ::/0
    Endpoint             = vpn_A.example.com:51820
```

user-3

```
[Interface]          # user-3 laptop
    PrivateKey           = <privkey>
    Address              = 10.77.77.4/32, fc00:77:77::4/128
    DNS                  = 10.10.10.10

#
# Gateways
#

[Peer]               # servers wg-A (gateway)
    PublicKey            = <pubkey wg-A>
    PresharedKey         = <psk wg-A x user-3.laptop>
    AllowedIPs           = 10.77.77.1/32, 192.168.1.32/28, 192.168.2.0/24
    AllowedIPs           = fc00:77:77::1/128
    Endpoint             = vpn_A.example.com:51820
```

CHAPTER

SIX

EDITING AND MAKING CHANGES

There are 2 classes of things that can be modified: the vpn information itself and any profile. Changes are made by using *–edit* option which writes a file. After any changes to that file they can be merged back using *–merge* option.

Files to be edited are written into files in the *Edits* directory. *Edits* is located in the working directory. Files use TOML format.

TOML, loosely speaking, has *key = value* pairs. Value can be an array which is simply comma separated list of items inside square brackets *x = [item1, item2, …]*. It can also have named sections which are started with [name]. Everythin following a named section belongs to that section until a new section is encountered. For additional infomation please see TOML documentation https://toml.io/en.

Each peer is identified by it's *ID* which is 3 tuple : *<vpn-name>.<account-name>.<profile-name>*. When editing the vpn itself, the *ID* is just the *<vpn-name>*.

Note that in order to rename something, then please see the *–rename* command line option which handles this.

6.1 Modifying Vpn Information

For example, to modify a vpn named *vpn-test*.

The first step is to make the edit request:

```
wg-tool --edit vpn-test
```

This creates a file under the *Edits* directory and the filename is displayed. In this example the file will be *Edits/vpn-test-info.mods*. The file is in standard TOML format.

You can then edit the contents of the file that contains the following items:

```
name = "vpn-test"
tag = "... do not change this ... "
dns = [
     "dns.exmple.com",
     "dns.example.net",
 ]
 dns_search = []
 peer_to_peer = false
 networks = [
     "10.77.77.0/24",
     "fc00:77:77::/64",
 ]
```

Never modify the *tag* as it is a unique identifier that determines where to merge any changes.

Most fields are self explanatory. If this is a new vpn then it is fine to change the networks but be cautious touching this once peers have been added. The code will refuse to remove a network if there are existing profiles with IP address assigned from it.

The peer_to_peer flag is discussed in a little more detail below.

After any changes are made the file they can be merged back using:

```
wg-tool -merge <filename>
```

The will update the config files with support for peer to peer communications.

6.1.1 Note On DNS

We use two variables to capture dns information. *dns* which is the list of dns servers and *dns_search* is the list of dns search domains.

Each host in *dns* may be an IP address or a hostname. Any hostname will use local DNS to resolve this to an IP address before they are output to the wireguard configs.

This applies not only to the vpn info, but equally to dns information used in profiles.

The reason for this, is that wireguard uses it's *DNS* variable for both dns servers as well as search dommains. It distinguises between them by recognizing IP addresses.

In a wireguard config, each element of *DNS* will is treated as a dns server only if it is a valid IP address (IPv4 or IPv6). Everything else is treated as a search domain.

6.1.2 Peer to Peer

If the *peer_to_peer* flag set to *true* then peers are permitted to communicate with one another using tunnel IP addresses. By default this is turned off and peers can only communicate with gateways. This can be changed at any time.

The gateway will need to have IP forwarding turned on. Standard way to do this in linux:

```
# /etc/sysctl.d/20-wireguard.conf
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1

# manually turn on if not already
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all.forwarding
```

Peer to peer can be turned off at any time, simply edit as above and set *peer_to_peer* = *false* and *merge* the change as usual.

6.2 Modifying Profile Information

Any profile, gateway or client can be modified by edting it's ID. The process is the same as for the vpn above using *edit/merge* but now using the profile ID.

We illustrate for the peer *vpn-test.user-3.laptop*.

```
wg-tool --edit vpn-test.user-3.laptop
```

Again, a file is created which you can edit and merge. In this example the file is *Edits/vpn-test.user-3.laptop.mods* which contains, aside from some comments at the top:

```
PersistentKeepalive = 0
MTU = ""
post_up = []
post_down = []
nets_offered = []
nets_wanted = []
internet_offered = false
internet_wanted = false
Endpoint = ""
Endpoint_alt = ""
dns = [ "dns.example.com", ]
dns_search = []
dns_lookup_ipv6 = false
dns_linux = false
use_vpn_dns = true
active = true
hidden = false

# change to tag = xxx
# keep id_str ??
[ident]
tag = "af4f6d5e-ae66-4bd1-afec-a7037cdd1d9e"
```

Once again, do not modify the tag. It is a unique identifier used to ensure edits go to the correct place.

If you need to rename an account or profile, then the *–rename* option should be used.

Most of the other items can be modified and are largely self-explanatory. Where appropriate we quote from the wireguard documentation.

- no_psk_tags

  This is a legacy tag used as part of migration only. It allows old migrated profiles that were not using pre-shared keys. Leave this one alone.

- PersistentKeepalive

  Per the wg docs: it is the seconds interval, between 1 and 65535, indicating how often to send an authenticated empty packet to the peer for the purpose of keeping a stateful firewall or NAT mapping valid persistently.

  Usually not needed, but if set 25 seconds is a sensible value.

  A good rule of thumb is if this peer only connects to other peers then this is unncessary. However when the peer needs to be available to be connected to, then it can be helpful to set this. For example, if this peer starts wireguard and sits behind one or more NAT routers, and is available for ssh to login, then setting keep alive will ensure there is a viable pathway from this peer to the gateway. This will make sure that you can ssh in to this peer.

  See *man wg* for more information.

- MTU

  If not specified, the MTU is automatically determined from the endpoint addresses or the system default route, which is usually a sane choice. However, to manually specify an MTU to override this automatic discovery, this value may be specified explicitly.

- post_up / post_down

  List of scripts to be run after setting up to tearing down the vpn interface.

  Gateway example:

```
post_up = ['/usr/bin/nft -f /etc/wireguard/scripts/postup.nft']
post_down = ['/usr/bin/nft flush ruleset']
```

- nets_offered / nets_wanted

  If this peer offers access to one or more networks, aside from internet, they can be listed here.

  Example providing access to a local network:

  nets_offered = ['192.168.1.0/24']

  Similarly if a peer wants to gain access they it lists the networks in net_wanted.

- internet_offered / internet_wanted

  Gateways may allow clients to have their internet traffic flow via the gateway. If that is allowed, then set *internet_offfered = true*.

  If a client wants use a gateway for all it's internet traffic, then it sets *internet_wanted = true*

  These are mutually exclusive, and gatways ignore internet_wanted, while clients ignore internet_offered.

- Endpoint

  Gatways use this to provide their host:ip. For example:

  Endpoint = "vpn.example.com:51820"

- Endpoint_alt

  A gateway may be available on and alternate endpoint. Typically this is an internal host or IP and this is directed at admin wishing to test the server on the internal network. With this added, you may then mark a profile with *alternate_wanted = true* and an separate wireguard config will written to the "Alt" subdirectory. e.g. If a user has a desktop profile so marked, then 2 configs will now be written instead of one. The only difference is the Endpoint used for the gateway.

```
Data-wg/vpn-test/user-1/desktop.conf
Data-wg/vpn-test/user-1/Alt/desktop.conf
```

- dns / use_vpn_dns / dns_lookup_ipv6 / dns_linux / dns_search

  DNS servers are set using *dns* variable and search domains using *dns_search*. DNS servers list may contain hostanmes or IP addresses (IPv4 or IPv6). They can be set in:

  - vpn info : *dns* variable
  - gateways : *dns* variable.
  - client's own *dns* variable.

  Dns servers are listed in order:

  current profile -> gateways -> vpn info

  DNS servers given has hostnames use DNS query to find their associated IP address(es) which are written the the wireguard configs.

  The reason is that Wireguard overloads it's *DNS* variable for both servers and search domains. Any item that is a valid IP is treated as DNS server and any that are not are used for DNS search domains.

  When *wg-tool* does a DNS query to find the IP address(es) for a hostname it uses all IPv4 addresses returned from the query.

If the *dns_lookup_ipv6* is *true* then it attemps to lookup any IPv6 addresses in addition. By default dns_lookup_ipv6 is *false*. While turning this on is benign on an IPV4 only machine, it can add delay in dns resolution due to IPv6 failure followed by IPv4 success.

Client profiles may toggle the flag *use_vpn_dns* = *false*, in which case they do not use dns servers provided by gateways or vpn info.

dns = ["dns.example.com",]

When *dns_linux* is not set, then the list of dns servers will be written to wireguard config [Interface] section.

If a linux client has *dns_linux* = *false*, then the resolv.conf will be managed by resolvconf (See man wg-quick for more info).

Use *dns_linux* = *true* to activate the DNS helper scripts for linux clients.

These use the dns servers together with any dns search domains as arguments to the helper script which is written to the postup/postdown variables.

dns_search follows the same inclusion logic as dns servers: profile -> gateways -> vpn info. Note that *dns_search* is only available for linux clients using the dns helper script.

The dns script should be installed in */etc/wireguard/scripts/wg-peer-updn*.

Please see *Linux Client DNS Script* for more details.

Any linux client using the script will have the dns servers and dns domain search set as arguments and written to the resulting wireguard config *postup / postdown* variable.

The DNS servers and domain search list are specified in vpn info and may be modified with *wg-tool –edit <vpn-name>*.

- active / hidden

  Profile may be marked active, inactive or hidden. Inactive profiles remain in the database and visible with *–list* but they are excluded from generating wireguard configs (*Data-wg*). hidden are treated as inactive but do not show with *–list* option unless *-verb* is also used. These states may be set during editing of a profile, or using the commmand line options. Command line options are helpful for quick changes or for changes to more than one profile.

  See *Command Line Options*.

- [ident]

  Dont touch this section which only contains a *tag*. This is a unique identifier and is used to ensure that any modifications go to the correct place.

  ID names can be changed using the *–rename* option and copies of profiles can be made with the *–copy-from* options.

  See the *Command Line Options* for more details.

CHAPTER

SEVEN

COMMAND LINE OPTIONS

As usual the help message is available via:

```
wg-tool --help
```

Which prints a list of available options.

Some items may be modified using *–edit* and some, like *–rename* have not edit counterpart.

Sometimes it may be more convenient to use an option rather than edit.

7.1 Current Help

```
usage: wg-tool [-h]
               [-wkd DIR]
               [-migrate]
               [-import-configs <vpn-name>]
               [-edit]
               [-copy]
               [-rename]
               [-ident vpn.account.prof]
               [-to-ident vpn.account.prof]
               [-merge FILE]
               [-nets-wanted-add NETS_WANTED_ADD]
               [-nets-wanted-del NETS_WANTED_DEL]
               [-nets-offered-add NETS_OFFERED_ADD]
               [-nets-offered-del NETS_OFFERED_DEL]
               [-new]
               [-roll]
               [-active]
               [-not-active]
               [-hidden]
               [-not-hidden]
               [-l]
               [-rpt RPTFILE]
               [-rrpt]
               [-b]
               [-r]
               [-fp]
               [-v]
               [-V]
```

(continues on next page)

(continued from previous page)

```
               [-hist NUM]
               [-hist-wg NUM]
               [id_names ...]

wg-tool: Manage wireguard config files

options:
  -h, --help            show this help message and exit
  -wkd, --work-dir DIR  Set the working directory path (./)

:*--------------------*:
: Positional Arguments :
:*--------------------*:
  One or more identity names ID(s)

  id_names              <vpn>.<account>.<profile> ...
                        e.g. client :       vpn1.bob.laptop
                        e.g. gateway:       vpn1.servers:gateway-1
                        NB Some options may take <vpn>.<account> or just <vpn>

:*------------------------*:
: Migrate & Import Options :
:*------------------------*:
  Migrating from versions prior to 8.0.

  -migrate, --migrate   Migrate database from pre v8.0 version:
                        configs -> Data, wg-configs -> Data-wg
                        Legacy directories are unchanged
  -import-configs, --import-configs <vpn-name>
                        Import standard wireguard configs.
                         The vpn must exist with matching and networks
                         All configs should be in the work dir and
                         the path to each config file should be:
                             <vpn-name>/<account-name>/<profile-name>.conf

:*---------------------*:
: Edit / Modify Options :
:*---------------------*:
  Add, edit or modify identities.

  -edit, --edit         Edit one item.
                        Requires --ident "vpn.account.profile"
  -copy, --copy         Copy one identity to another.
                        Requires --ident <from> --to-ident <to>
  -rename, --rename     Rename an identity.
                        Requires --ident and --to-ident
  -ident, --ident vpn.account.prof
                        Specify an identity.
                        Use with --edit, --copy and --rename)
  -to-ident, --to-ident vpn.account.prof
                        Specify target identity,
                        Use with: --copy and --rename)
```

(continues on next page)

(continued from previous page)

```
  -merge, --merge FILE  Merge edits from a file.
  -nets-wanted-add, --nets-wanted-add NETS_WANTED_ADD
                        Add wanted network(s) to IDs (See positional parameters)
                        Multiple networks use comma separated list
                        Special network "internet" is same as internet_wanted = true
  -nets-wanted-del, --nets-wanted-del NETS_WANTED_DEL
                        Delete wanted network(s) from IDs (See positional parameters)
                        Multiple networks as comma separated list
                        Special network "internet" is same as internet_wanted = false
  -nets-offered-add, --nets-offered-add NETS_OFFERED_ADD
                        Add offered network(s) to IDs (See positional parameters)
                        Multiple networks use comma separated list
                        Special network "internet" is same as internet_offered = true
  -nets-offered-del, --nets-offered-del NETS_OFFERED_DEL
                        Delete offered network(s) from IDs (See positional parameters)
                        Multiple networks as comma separated list
                        Special network "internet" is same as internet_offered = false
  -new, --new           Create new item (See positional parameters))]
                        Each can be one of: vpn, vpn.account or vpn.account.prof
  -roll, --roll-keys    Generate new keys for IDs (See positional parameters)
  -active, --active     Mark some IDs active. (Use positional parameters)
  -not-active, --not-active
                        Mark some IDs not active. (See positional parameters)
  -hidden, --hidden     Hidden are not shown unless double verb -vv is used.
  -not-hidden, --not-hidden
                        Mark some IDs not hidden. (See positional parameters)

:*-----------------*:
: Reporting Options :
:*-----------------*:
  Peer info and reports from wg server report

  -l, --list            List vpns, accounts & profiles. May filter by providing IDs
                        profile names may comma separated list
                        e.g.
                          all : -l
                          all accounts and profiles of "vpn1": -l vpn1
                          all profiles of "vpn1.bob": -l vpn1.bob
                          2 profiles of "vpn1.bob": -l vpn1.bob.prof-1,prof-2
  -rpt, --show-rpt RPTFILE
                        Generate report from "wg show" output (file or "stdin") (See␣
↪also --run-show-report)
  -rrpt, --run-show-rpt
                        Run "wg show" (must be on wireguard server) and generate␣
↪report. (See also --show-rpt)
  -b, --brief           Simplified outputs. Opposite of -det

:*---------------*:
: General Options :
:*---------------*:
  Miscellaneous options.
```

(continues on next page)

(continued from previous page)

```
  -r, --refresh         Force a refresh and write all wireguard configs.
                        This is automatic and is not needed normally.
  -fp, --file-perms     Redo restricted file permissions on all data files. Not␣
↪normally needed.
  -v, --verb            Verbose output. Repeat for even more verbosity.
                        -vv works with -l and -rpt. Use -vvv for very verbose output
  -V, --version         Version info

:*--------------*:
: Stored Options :
:*--------------*:
  These are saved as new default values.

  -hist, --hist NUM     Number of previous data configs to retain (5)
  -hist-wg, --hist-wg NUM
                        Number of previous wireguard configs to retain (3)
```

7.2 Compacting Networks

Note on networks. The wireguard config *AllowedIPs* variable compacts networks to the minimal list of CIDRs. For convenience, the pre-compacted list of networks is provided as a comment.

For example:

```
[10.77.77.1/32, 0.0.0.0/0, 192.168.1.0/24, ::/0, fc00:77:77::1/128]

gets compacted to →

[0.0.0.0/0, ::/0]
```

since *192.168.1.0/24* is a subnet of *0.0.0.0/0* and similarly for the IPv6 net.

A comment is added to the config file which shows all the *AllowedIPs* networks prior to compacting. This can be helpful when looking them over for testing or checking purposes.

CHAPTER

EIGHT

REPORTS

8.1 A Better Gateway Report.

The *-rpt* option reads output from the standard wireguard gateway report generated by *wg show* and presents it in a human friendly way.

The *wg show* output uses public keys to identify each peer. *wg-tool* looks up which profile the key belongs to which enables it to show the same information by account and profile name.

To use this save the output of *wg show* into a file and then run

```
wg-tool -rpt <filename>
```

This feature solves a long standing problem with native wireguard reports which burden the administrator with mapping IPs or public keys to an account.profile.

The report also indicates if accounts / profiles seen by the gateway are active (or inactive) by showing (+) or (-) beside the name.

If an inactive profile is seen on the gateway, it may be time update the gateway configs.

Because of this feature, this tool eliminates any need for complex schemes, some are pretty dodgy. For example, so-called Vanity keys, which attempt to search for public keys that are more palatable by repeatedly generating keys until one is found that looks it might contain a sequence of characters that is *interesting*.

8.2 Sample Report

This is a sample output taken from *wg show*:

```
interface: wg0
  public key: AAAAA...
  private key: (hidden)
  listening port: NNNNN

peer: XXXX...
  preshared key: (hidden)
  endpoint: x.x.x.x:nnnnn
  allowed ips: 10.77.77.134/32
  latest handshake: 11 seconds ago
  transfer: 21.57 KiB received, 120.62 KiB sent
```

And this is the report generated from that same output using *wg-tool -rpt <report file>*:

```
vpn: vpn1 (+)

servers (+)    : wg-A (+) (gateway)
     interface : wg0
  listen_port  : NNNNN
      address  : ['10.77.77.1/24', 'fc00:77:77::1/64']

Peers using this gateway
alice (+)      : laptop (+)
      address  : '10.77.77.134/32'
   allowed_ips : '10.77.77.134/32'
      endpoint : x.x.x.x:nnnnn
      transfer : 21.57 KiB received, 120.62 KiB sent
```

A little more detail is shown when coupled with the *-det* option:

```
  vpn: vpn1 (+)

  servers (+)    : wg-A (+) (gateway)
       interface : wg0
    listen_port  : NNNNN
        address  : ['10.77.77.1/24', 'fc00:77:77::1/64']
internet_offered : True
       PublicKey : AAA...

  Peers using this gateway
  alice (+)      : laptop (+)
        address  : '10.77.77.134/32'
     allowed_ips : '10.77.77.134/32'
        endpoint : x.x.x.x:nnnnn
        transfer : 21.57 KiB received, 120.62 KiB sent
 internet_wanted : True
       PublicKey : XXX...
```

CHAPTER

NINE

KEY ROLLOVER

9.1 Generating New Crypto Keys

wg-tool makes key rollover particularly simple - at least as far as updating keys and regenerating user and/or server configs with the new keys.

Each peer utilizes one *private/public key pair*, and the *public* key is shared with all any peers that it communicates with.

In addition every pair of peers that communicate with one another are assigned a unique pre-shared key (or *PSK*). The *PSK* strengthens the security of the communications between that pair and in particular enhances post-quantum resistance.

When new keys are generated, the peer gets a new *private/public* key pair. In addition any needed *PSKs* are re-generated.

As usual, after new keys are made, the corresponding wireguard configs are automatically updated. This includes any impacted by a change in *PSKs*.

Distribution of the updated configs and QR codes to each user is not addressed by the tool. Continue to use existing methods - encyrpted email, in person display of QR code etc.

Its as simple to update gateway or client keys - just specify those to change on command line.

To roll the server keys run:

```
wg-tool --roll-keys <id or id list>
```

Once this has completed, any affected wireguard configs will be updated appropriately.

CHAPTER

TEN

CONVENIENCE SCRIPTS

10.1 Gateway Nftables Script

The nftables sample script is found in:

```
scripts/postup.nft
```

it should be copied to /etc/wireguard/scripts on each gateway server.

Edit the network variables at the top of the *postup.nft* script to match the server network settings.

In the event a gateway provides client access to the internet via the gateway, and / or to any internal network(s) this script will NAT all tunnel traffic providing access to the same networks the server has access to.

The *postup.nft* script provides access to the internet and lan provided the wireguard server host has that access.

If the gateway is behind a border firewall then that must forward the vpn (UDP) traffic to the wireguard server which running on the inside.

If the gateway server is in a DMZ then it may have more limited access.

Before deploying the *postup.nft* script, edit the 3 variables at the top for each server setup it is being deployed on:

- vpn_net

  this cidr block must match the wireguard internal tunnel network.

- lan_ip lan_iface

  IP and interface of the server

Remember also to allow forwarding on the gateway server, to ensure VPN traffic is permitted to go to the LAN:

```
sysctl -w net.ipv4.ip_forward=1
```

to keep this permanent, add to */etc/sysctl.d/20-sysctl.conf* (or similar filename):

```
net.ipv4.ip_forward = 1
```

The gateway config generated by the tool is how peers are managed. For example existing Accounts/profiles can be marked inactive to remove them from the gateway config and new ones can easily be added. After making changes, simply copy the resulting gateway config to /etc/wireguard/ where you can use *wg-quick* to start/stop the wireguard server.

10.1.1 Linux Client DNS Script

This is located in:

```
scripts/wg-peer-updn
```

When using linux as a wireguard client, there are 2 ways to bring up wg.

- by default wg-quick uses resolvconf to change /etc/resolv.conf If using this method this script is not needed. The client.conf file should have DNS = comma separated list of IPS and optinally domains to add to search list.
- PostUp to save existing resolv.conf, install wg resolv.conf PostDown to restire the resolv.conf from saved.

This script implements the second approach and handles both –up and –down

e.g. PostUp = wg-peer-updn -u -dns 10.0.0.1,10.0.0.2 -dnsrch foo.example.com,example.com PostDown = wg-peer-updn -d

In either case to start and stop vpn

wg-quick up <foo.conf> wg-quick down <foo.conf>

In practice, for any linux client simply copy the script on the client:

```
cp scripts/wg-peer-updn /etc/wireguard/scripts/wg-peer-updn
```

and *–edit* the client profile and set the flag:

```
dns_linux = true
```

wqg-tool will take care of the rest. It will compile the unique list of DNS servers taken from profile, all gateways and vpn info.

This will use whatever DNS servers the gateway provides (via *DNS* setting) together with the script in the client profile's *PostUp/PostDown* settings.

Note that on other operating systems (e.g. phones), they use whatever DNS servers the gateway provides in it's *dns* variable.

See *Command Line Options* for more detail.

CHAPTER

ELEVEN

MIGRATING FROM EARLIER VERSION

Version *8.x* introduces significant changes including with the database file formats. This section discusses migration from earlier versions of *wg-tool* to the current version.

If you have not used *wg-tool* and would like to import existing standard wireguard configs, then see *Importing From Standard Wireguard Configs* instead.

Migration from an earlier version is pretty much automatic.

In the earlier versions gateways were known as servers and accounts were called users.

In current version each item has an identity comprised of 3 parts:

- vpn name
- account name
- profile name

And would be denoted by, for example: *vpn1.alice.laptop*. The directory file hierarchy follows this.

N.B.

- All existing data is untouched.
- New data lives in different directories.

After migration, the older (pre *8.x*) directories (*configs* and *wg-configs*) may be deleted when convenient.

All tool data now resides under *Data* and standard wireguard configs are output under *Data-wg*.

To migrate existing data, run:

```
wg-tool --migrate
```

All being well this should populate both *Data* and *Data-wg*. You can examine the wireguard configs under *Data-wg* and you can see a list of all accounts and their profiles using:

```
wg-tool --list
```

11.1 Gateway with Alternate Endpoint

If you had an alternate endpoint in the gateway profile it will be now be in the the *Endpoint_alt* field.

If you had any clients using the alternate endpoint then you can now mark a profile using *alternate_wanted = true*. When this is set for some profile, then a second wireguard config will be created using the alternate endpoint. That config is identical to other than the Endpoint.

For any such profiles you'll need to do this step manually after the migration. In order to change a profile this way you use the *–edit* option.

For more detail please see *Editing and Making Changes*.

```
wg-tool --edit vpn1.alice.laptop
```

Which creates a regular file and displays the filename. In this example it would be

```
Edits/vpn1.alice.laptop.mods
```

You then edit the file with any text editor and change

```
*alternate_wanted = true*.
```

The change is merged back using:

```
wg-tool --merge Edits/vpn1.alice.laptop.mods
```

At this point any profile with *alternate_wanted = true* will have a new config saved in the "Alt" subdirectory where the profile is.

For example, here we would find:

```
Data-wg/vpnt/alice/laptop.conf
Data-wg/vpnt/alice/Alt/laptop-alt.conf
```

and the associated QR code image file under *qr* directory.

After this is done, you may want to mark any older profiles using this altenate Endpoint as *hidden*, since they are no longer needed.

```
wg-tool --hidden vpn1.alice.<no-longer-needed>
```

Where the *<no-longer-needed>* would be replaced by whatever profile name is not needed.

11.1.1 Using DNS Instead

If an alternate endpoint uses a domain name, then the best approach is to have the local DNS server map that host name to it's internal IP address when seen on the internal network, and the usual public IP when on the outside.

This way there is no need for any additional client configs at all. Having clients use *alternate_wanted* is really only useful when the local DNS option is not available.

11.2 Hiding Unused Profiles

For all available options run *wg-tool –help* or see the section *Command Line Options*.

Since this may come up after migrating, a brief mention of this functionality may be useful.

If you you have profiles that are no longer needed they can be marked *not-active* or *hidden*. In both cases, wireguard configs are no longer generated and all data is retained in case of need in the future.

If a profile is marked hidden then it remains hidden with *–list* unless *–verb* is also used. If it's marked not active, then it will always be visible when listing profiles *–list*. Profiles that are not active or hidden do not get a wireguard config generated.

for example:

```
wg-tool --hidden vpn1.alice.laptop
```

The active/hidden states may also be changed when *editing* a profile.

CHAPTER

TWELVE

IMPORTING FROM STANDARD WIREGUARD CONFIGS

It is possible to import from standard wireguard configs. This makes it straightforwd to move an existing vpn setup so it can be managed by *wg-tool*.

Note that there is one caveat which may require small adjustment after import completes.

12.1 Limitations and Networks

Wireguard configs provide network information in the *AllowedIPs* variable. *wg-tool* requires some knowledge about the network topology to enable it to make appropriate adjustments as any changes are made.

Specifically networks are treated as provided by (*nets_offered*) or desired (*nets_wanted*). These can be modified by editing *nets_offered* or by using the command line option *nets-wanted-add*. And similarly for *wanted*. The wireguard configs do not relate the network topology, only what each pair of peers is willing to communicate. Which is fine and complete for it's needs.

Hence, after importing, you may need to manually add the right information about which peers offer or want which networks.

As simple example consider a vpn with a single gateway and one client sharing a single network *net-1*. From the configs all we know is that this pair of peers shares *net-1*. We cannot tell which of the 2 actually is routing the network to the other.

The resulting wireguard configs generated by *wg-tool* after importing will match those that were imported.

If nothing changes, it doesn't really matter either. While importing these configs *wg-tool* we will simply assign the owner of *net1* to (our *nets_offered* varable) to the gateway. Regardless of which of these 2 peers actually is proving *net1* access.

Lets say it is actually the client that provides the network access not the gateway.

Let's see what happens if we don't correct this and leave the gateway as the owner.

Now imagine what happens if the client here, the real provider of access to *net1*, is made inactive. At this point the gateway is still claiming to provide access to *net1* for it's other clients and this is of course is not true.

Bottom line, after importing, run *wg-tool -lv* to get a listing of all peers and which offer networks, and correct any that are incorrectly guessed.

For example if *10.1.2.0/24* was incorrecly set to peer vpn.serv.A instead of vpn.serv.B then you can correct this using:

```
wg-tool --net-offered-del '10.1.2.0/24' vpn.serv.A
wg-tool --net-offered-add '10.1.2.0/24' vpn.serv.B
```

You can of course use the *–edit* option to make changes as well.

It's a good idea to run *wg-tool -lv* after these changes so you can check things are as expected.

If we import the wireguard configs from where we left Example 3, then *wg-B* offer both networks instead of offering *192.168.2.0/24* and wanting *192.168.1.0/24*.

Furthermore, user-1 and user-2 show *internet* wanted which is strictly correct though it was originally created with wanting both *192.168.1.0/24* and *192.168.2.0/24* as well as internet.

Bottom line, after importing always manually clean up *nets_wanted* and *nets_offered*.

Back to importing.

12.2 Step 1: Create New VPN

Before importing any configs you must create the vpn along with some basic information about the vpn. Ths is to establish the vpn name, networks and dns hosts.

You **must** establish the vpn networks that match those used in the wireguard configs about to be imported.

To illustrate we'll create a new vpn called 'vpn-imported' with tunnel networks:

```
vpn_nets = ['10.77.77.0/24', 'fc00:77.77::/64']
```

This assumes that all configs about to be imported are using a tunnel IP address from those in *vpn_nets*.

This is best accomplished working in a fresh directory.

```
wg-tool -wkd ./
wg-tool --new vpn2
```

Now edit the vpn info file, name will be shown (*./Edits/vpn2-info.mods*) and set the desired vpn networks and any DNS servers.

```
dns = ["10.10.10.10"]
peer_to_peer = false
networks = [
     "10.10.10.0/24",
     "fc00:10.10::/64",
]
```

Then merge these changes back:

```
wg-tool --merge ./Edits/vpn2-info.mods
```

12.3 Step 2: Import Standard Wireguard Configs into VPN

wg-tool uses a ID string taking the form:

```
<vpn-name>.<account-name>.<profile-name>
```

Import uses directory and file names that maps to such an ID.

Create the directory where all the wireguard configs will be imported from. The top level directory name should be the vpn-name chosen above.

```
mkdir vpn2
```

With the vpn details established all standard wireguards configs should be put into under this *vpn2* directory.

Directly under *vpn2* will be an *account* directory for each account. Each account has it's profile config(s) in it's directory.

In other words, each config file should have it's file path end with *vpn2/<account-name>/<profile-name>.conf*.

which tells the tool what ID is to be used for that config.

For example, the config file to be imported into *vpn-imported.sally.laptop* should be placed in:

```
vpn-imported/sally/laptop.conf
```

Do this for all gateway and client configs to be imported.

For any config that runs on a server, such as gateways or clients that provide LAN access, we like to keep all of those under a *servers* account. Choose whatever names you prefer but characters used should be alphanumeric along with any of "- = _ : , ~ + ;".

Once all the configs are in place, then we're ready to start the import.

```
wg-tool --import-configs vpn2
```

This means the vpn name is *vpn2* and configs are imported from the within the *vpn2* directory.

All being well, you can check the newly generated configs run *wg-tool -lv* to list the current state and look compare the newly generated configs under *Data-wg/vpn2* with those that were imported from *vpn2/*

It's good to confirm the new configs are correct. In particular keys (public, private and pre-shared), along with AllowedIPs and Endpoints.

As discussed earlier, if needed, correct any networks being shared such that the profile responsible for access to any network, has it so listed under it's *nets_offered* variable and removed from any profile that is not the actual access provider.

Likewise, especially if you made any changes to *nets_offered* you may need to adjust *nets_wanted*.

If an error is encountered during the import, it should display a message and exit.

CHAPTER

THIRTEEN

DIRECTORY AND FILE STRUCTURE

wg-tool uses files to hold all it's data. It uses 3 directories, one for it's own data, one for the wireguard config output and one for edits to existing data.

- *Data*

  This holds all the tool's own data. Every vpn, account and profile data reside under this.

- *Data-wg*

  The wireguard configs, which are the final output, are all under this.

- *Edits*

  This is used whenever any item is modifed. The *–edit* option saves a file here, which can be modified and merged back (*–merge*). It's just a temporary working area used while making any changes.

13.1 Tool Database and Wireguard Configs

Each individual item uses an identifier that is made of 3 parts:

```
vpn-name . peer-name . profile-name,
```

and the directory structure for both *Data* and *Data-wg* follows this hierarchy.

All the wireguard configs are saved in *Data-wg*. This includes any QR codes for client profiles which are in it's *qr* sub-directory.

```
Data/
    <vpn-name>/
        <account-name-1>/
                    <profle-name-1>.prof
                    <profle-name-1>.prof

        <account-name-2>/
                    <profle-name-1>.prof
                    <profle-name-1>.prof
```

```
Data-wg/
    <vpn-name>/
        <account-name-1>/
            <profle-name-1>.conf
            <profle-name-1>.conf
                    qr/
```

(continues on next page)

(continued from previous page)

```
                        <profile-name-1>.png
                        <profile-name-2>.png
        <account-name-2>/
            <profle-name-1>.conf
            <profle-name-1>.conf
```

For both Data and Data-wg files, each file is a symlink to a file which resides under the *_.db._* subdirectory. This permits keeping some history on each file. The number of history files kept is an available option. There are separate options for Data and Data-wg.

For example:

```
<some directory>
            laptop.prof -> _.db._/<date-1>/laptop.prof

            _.db._/
                <date-1>/laptop.prof
                <date-2>/laptop.prof
                ...
```

When *wg-tool –edit* **<ID>** is used it displays the edit file path. Changes to that file can be merged back with the *–merge* option.

All edit files fall under the *Edits* directory:

```
Edits/
     <vpn-name>-info.mods
     <vpn-name>:<account-name>:<profile-name>.mods
```

CHAPTER

FOURTEEN

APPENDIX

14.1 MTU Observation

I came across one hotel wifi where the vpn worked providing internet access, but using ssh was problematic.

```
ssh -v internal-host
```

would hang right after it logged:

```
expecting SSH2_MSG_KEX_ECDH_REPLY
```

The *fix/work around* was changing the MTU setting from 1500 down to 1400 on my laptop while at that hotel.

With that change everything worked normally again.

I have only ever had to modify the MTU at this one location, but I mention it in case someone else has a similar issue.

14.2 Install

While it is simplest to install from a package manager, manual installs are done as folllow:

First clone the repo :

```
git clone https://github.com/gene-git/wg_tool
```

Then install to local directory. When running as non-root then set root_dest to a user writable directory.

```
rm -f dist/*
/usr/bin/python -m build --wheel --no-isolation
root_dest="/"
./scripts/do-install $root_dest
```

14.2.1 Dependencies

Run Time :

- python (3.13 or later)
- wireguard-tools
- nftables (for wireguard server postup.nft)
- py-cidr
- python-qrcode

- tomli_w (aka python-tomli_w)

Building Package:

- git
- hatch (aka python-hatch)
- wheel (aka python-wheel)
- build (aka python-build)
- installer (aka python-installer)
- sphinx
- myst-parser
- texlive-latexextra (texlive tools, this is archlinux package)
- rsync

14.2.2 Philosophy

While we follow PEP-8, PEP-257, PEP-484 and PEP-561, we prefer to allow max line length of 100 rather than 79 in some situations.

We follow the *live at head commit* philosophy as recommended by Google's Abseil team[1]. This means we recommend using the latest commit on git master branch.

14.3 License

[1] https://abseil.io/about/philosophy#upgrade-support

CHAPTER

FIFTEEN

CONTRIB

15.1 Ubuntu Notes

Provided by Jack Duan (@jduan00 via github #13)

I would like to share the steps I got it to work on Ubuntu 22.04. Feel free to include in your documentation to help others.

15.1.1 Install Prep

Assume these are already installed.

```
$ sudo apt install wireguard qrencode
```

Install necessary python3 packages for wg_tool:

```
$ sudo apt install python3 python-is-python3 python3-poetry python3-build \
python3-installer python3-qrcode python3-tomli-w python3-tomli python3-netaddr
```

15.1.2 Build and install

```
$ git clone https://github.com/gene-git/wg_tool
$ cd wg_tool

$ sudo pip install hatchling

$ /usr/bin/python -m build --wheel --no-isolation

$ sudo ./scripts/do-install /
```

15.1.3 Getting wg and wg-tool to work

```
$ sudo -i
# cd /etc/wireguard

# wg-tool --init

# vi config/server/server.conf
```

Add the first set of users with wg-tool

```
# wg-tool --add_user --dns_search Mary:mac,iphone,ipad

# ln -s wg-configs/server/wg0.conf .

# systemctl enable wg-quick@wg0
# systemctl start wg-quick@wg0
# systemctl status wg-quick@wg0

# wg show
```

Add more users

```
# cd /etc/wireguard
# wg-tool --add_user --dns_search Joe:mac,iphone,ipad

# systemctl reload wg-quick@wg0

# wg show

# wg-tool --list_users

# wg-tool -rrpt
```

15.1.4 iptables

Also, I don't use PostUp scripts since persistent iptable rules are used.

```
# apt install iptables-persistent
# systemctl status iptables
# systemctl enable iptables
# systemctl restart iptables
```

Change /etc/iptables/rules.v4 Note

- 192.168.100.0/24 is used for wg subnet.
- On Ubuntu ens3 is the default network device name

```
# == For Wireguard VPN ==
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
# -- NAT for vpn clients from wireguard
-A POSTROUTING -o ens3 -s 192.168.100.0/24 -j MASQUERADE
COMMIT
# ----------------------
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo   -j ACCEPT
-A INPUT -i wg+  -j ACCEPT
```

(continues on next page)

(continued from previous page)

```
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m state --state INVALID             -j DROP
# -- Wireguard ports
-A INPUT -i ens3 -p  udp -m multiport --dports 51820 -m state --state NEW -m limit --
↪limit 10/sec --limit-burst 20 -j ACCEPT
-A INPUT -i ens3 -p icmp -m icmp --icmp-type 8 -m limit --limit 5/sec -j ACCEPT
# -- last as the default rule for input
-A INPUT -j DROP
# -- forward --
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -m state --state INVALID             -j DROP
-A FORWARD -i wg+  -s 192.168.100.0/24          -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-port-unreachable
# -- out --
-A OUTPUT -m state --state INVALID -j DROP
COMMIT
# == EOF ==
```

Run commands:

```
# systemctl restart iptables

# iptables -nvL
# iptables -t nat -nvL
```