Trusted Networks with WireGuard

Tom Ryder
tom@sanctum.geek.nz
https://sanctum.geek.nz/

- In principle, I **love** VPNs.

VPNs suck—2/4

- In principle, I **love** VPNs.
- In practice, I **hate** them.
  - Poor stability
  - Brittle and confusing configuration
  - Snake-oil “security” vendors
  - Proprietary software

VPNs suck—3/4

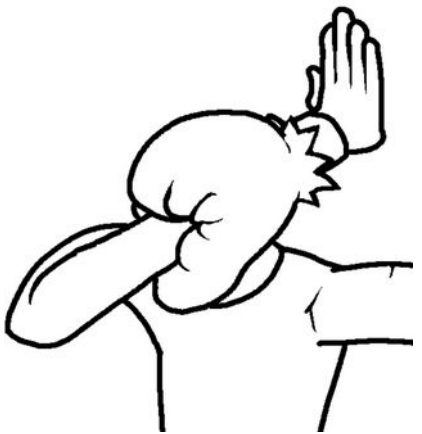
- Proprietary software for VPNs is *not acceptable* since Snowden.
- Your VPN *must* be free software, open for the public to audit and control.
  - This is not negotiable.
  - It’s as important as the endpoints’ operating systems.
- FortiClient VPN, Juniper Secure Connect, Sophos VPN …all rubbish.

VPNs suck—4/4

Just in case you still trust big security vendors to get this sort of thing right…

FortiGuard Used Hardcoded Key, XOR to Encrypt Communications

By **Ionut Ilascu** November 25, 2019



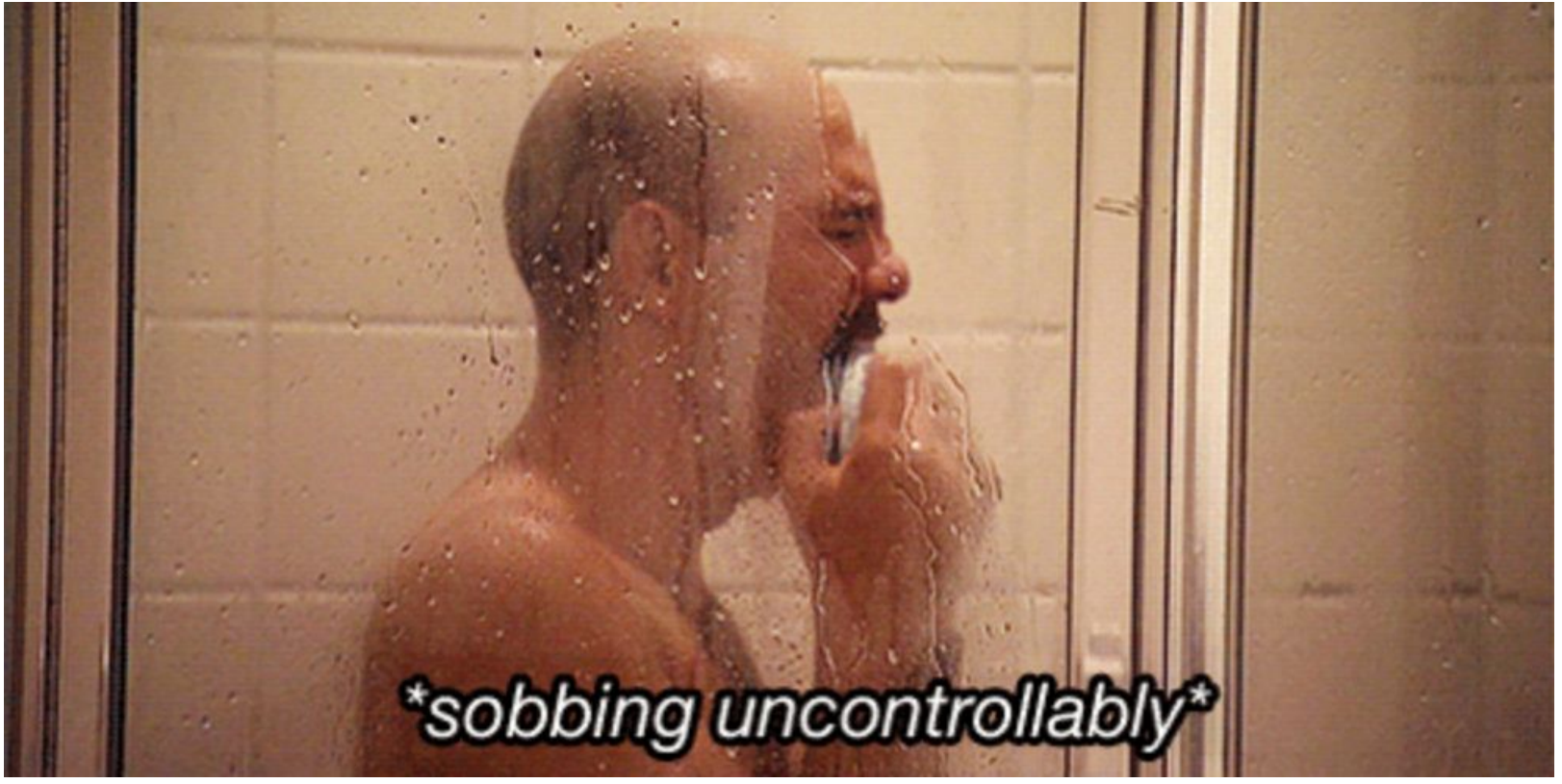
FortiGuard Used Hardcoded Key, XOR to Encrypt Communications—Bleeping Computer, November 25, 2019

VPN requirements

- Must be free software. No exceptions.
- Must be cryptographically sound (secure).
- Must not have too much overhead (fast).
- And—let's be honest…
  …must be *easy*…
  *…*or at least *straightforward*.

IPsec—1/2

IPsec—2/2



Good enough—1/2

OpenVPN ticks all the must-have boxes.

- Survived a long time as such: first released in **2001.**
- This talk *isn't* an OpenVPN bashing session. Thank you to the OpenVPN developers!

Good enough—2/2

- But setting up OpenVPN is a *pain*.
- The *correct* approach is still to roll your own **X.509 PKI Certificate Authority** for peer authentication!
- Yeah, you can do **symmetric pre-shared keys**…

  (but please don't…it's insecure, and it doesn't scale)
- Easy-RSA eases the pain a little.
  - If you're stuck on OpenVPN, or even need a TLS tunnel with an ad-hoc PKI anywhere else, give that a try.

Lean on me—1/4

So, I sinned; I just used **OpenSSH**.

- It's free software.
- It runs on *anything*, even Windows.
- It tunnels TCP traffic anywhere I want.
- It creates TCP *proxies* for me anywhere I want (SOCKS).
- I can mount network filesystems over it (`sshfs`).

Lean on me—2/4

- After all, authenticating peers with OpenSSH is *really* easy.
- Especially with the newer short **ed25519 keys**.

```
$ ssh-genkey -t ed25519
$ ssh-copy-id user@host
```

- If only a VPN could be that easy…

Lean on me—3/4

“When SSH is the foundation of your security architecture, you know things aren’t working as they should.”

—Rob Pike

Credit: Kevin Shockey

Lean on me—4/4

- It’s great that we *can* use SSH like this…
- …but we probably *shouldn’t.*
- It’s for a secure login **shell**, after all.
- It’s not a general-purpose network encryption tool.
- It’s certainly not a VPN.
- It’s big and complicated enough already.

Enter WireGuard

- We already have **Jason A. Donenfeld** to thank for:
  - cgit (CGI web frontend for Git)
  - pass (GNU Bash and GnuPG password manager)
- Now we have WireGuard, too.

Credit: ISRG

What is WireGuard?

- Layer 3 (IP) point-to-point VPN
- Copy-pasteable SSH-style public keys
- IPv4 and IPv6
- Works well with modern Linux features
  - Containers
  - Network namespaces
- Ported to other operating systems, too
- Lots of ports already (Rust, Go…)

Short and sweet

WireGuard's code is about

1%

of OpenVPN's in size.

(Not a typo!)

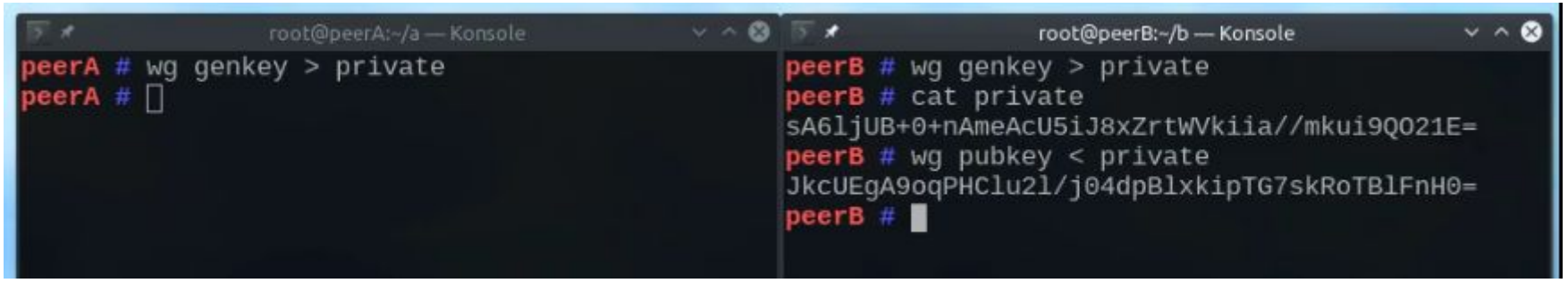
Demonstration—1/10

Please watch this screencast video first.
(video/mp4, 2m25s, 1.8 MiB)

(Archived version here.)

Tom will comment briefly as the video plays.
Don’t worry, we’ll go over it again
afterwards, slide-by-slide.

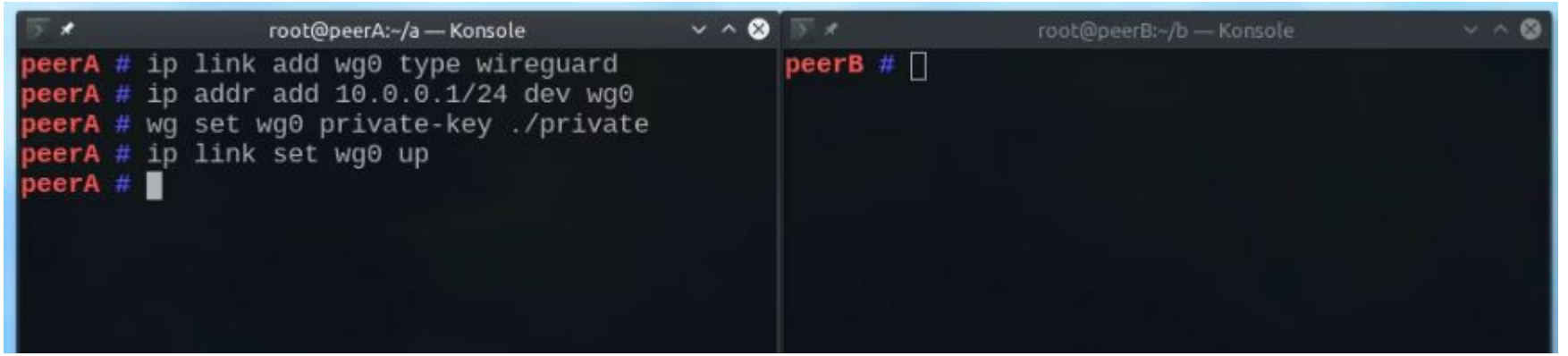
Demonstration—2/10



Generate Curve25519 keys

- Done with the generic `wg(8)` tool
- One key pair (private and public) on both peers
- Public key can be derived from private key
- Keys are represented in base64
- Short, copy-pasteable

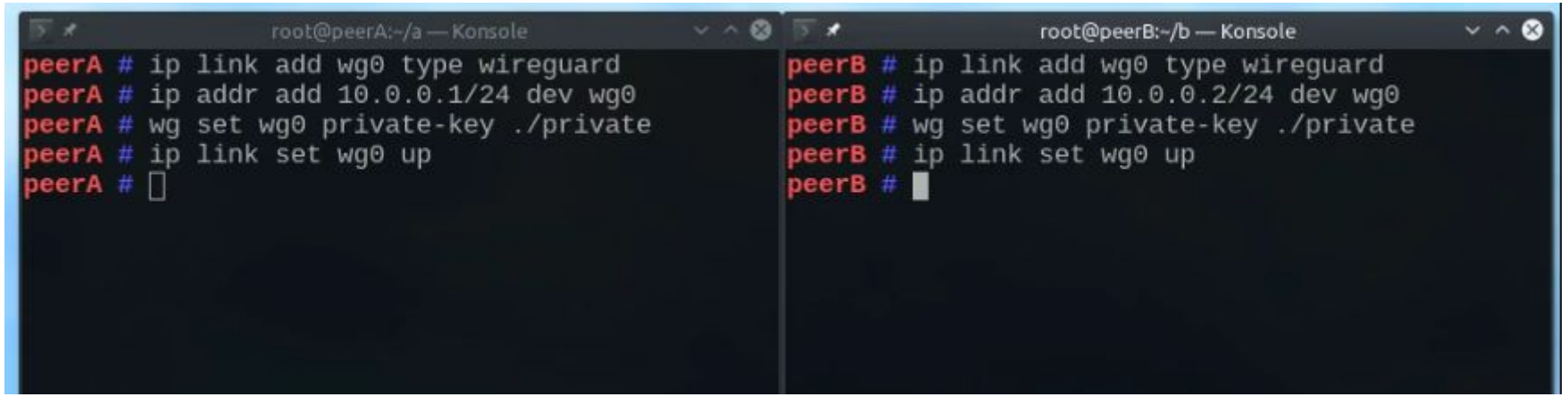
Demonstration—3/10



Create WireGuard interface wg0 on peerA

- “wireguard” is a valid network interface type to `ip(8)`
- Add an interface and an address 10.0.0.**1**/24
- Set the interface’s private key as created before
- Raise the interface

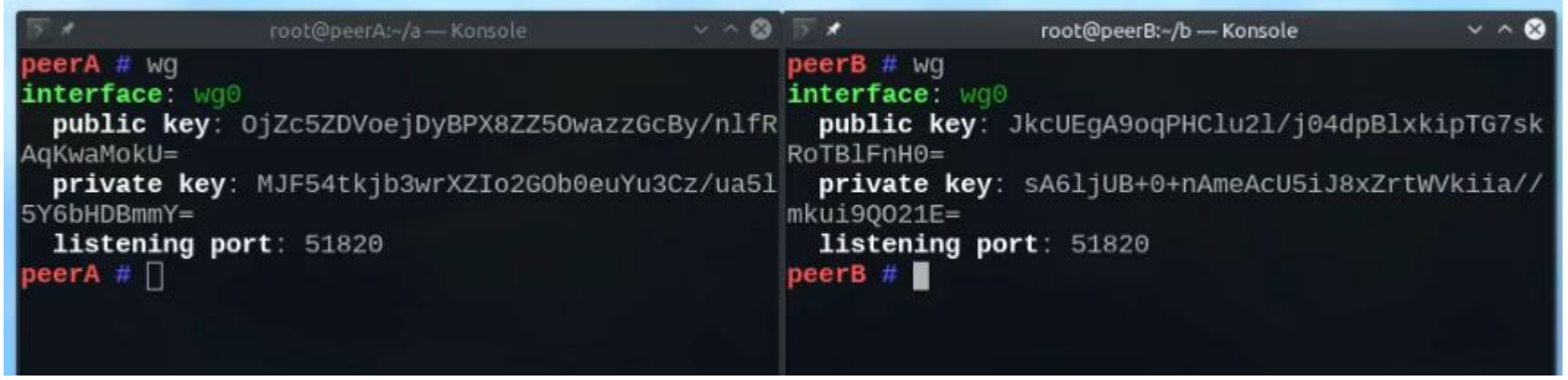
Demonstration—4/10



Create WireGuard interface wg0 on peerB

- Same again; add address 10.0.0.**2**/24

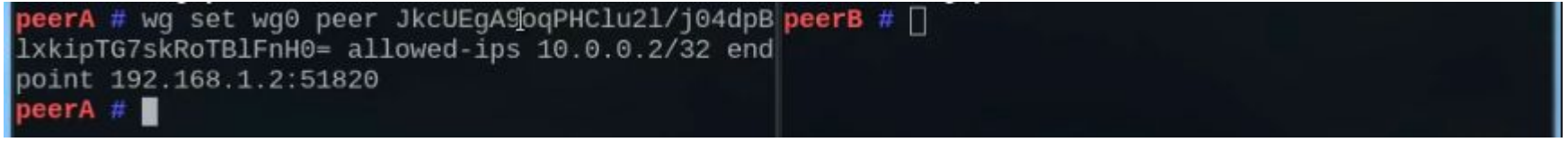
Demonstration—5/10



List key information on both peers

- Public and private keys on both
- Listening port (default udp/51820) on both

Demonstration—6/10



Add *peerB*'s public key and address to *peerA*

- The peer's key has to be known to WireGuard
- The peer may *only* use the configured addresses

Demonstration—7/10

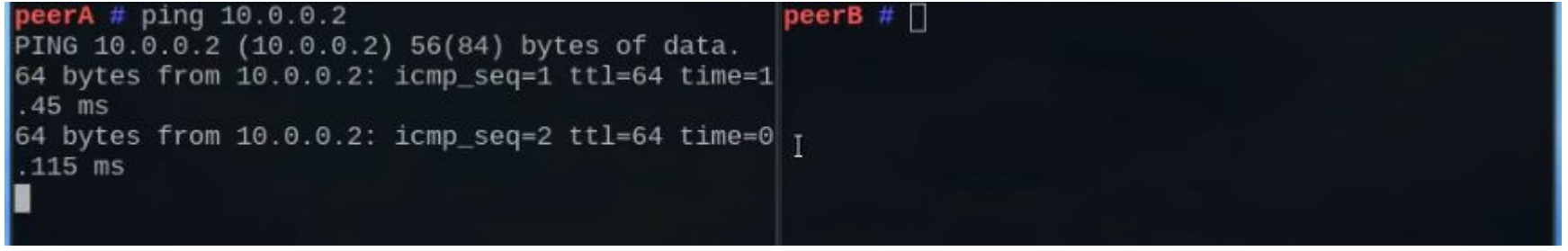
```
peerA # wg set wg0 peer JkcUEgA9oqPHClu2l/j04dpBlxkipTG7skRoTBlFnH0= allowed-ips 10.0.0.2/32 endpoint 192.168.1.2:51820
peerA #
```

```
peerB # wg set wg0 peer OjZc5ZDVoejDyBPX8ZZ5OwazzGcBy/nlfRAqKwaMokU= allowed-ips 10.0.0.1/32 endpoint 192.168.1.1:51820
peerB #
```

Add *peerA*'s public key and address to *peerB*

- Same again
- This is a peer-to-peer link, no “server” or “client”

Demonstration—8/10



We have ICMP ECHO (ping)!

Demonstration—9/10

```
peerA # wg
interface: wg0
  public key: OjZc5ZDVoejDyBPX8ZZ5OwazzGcBy/nlfR
AqKwaMokU=
  private key: MJF54tkjb3wrXZIo2GOb0euYu3Cz/ua5l
5Y6bHDBmmY=
  listening port: 51820

peer: JkcUEgA9oqPHClu2l/j04dpBlxkipTG7skRoTBlFnH
0=
  endpoint: 192.168.1.2:51820
  allowed ips: 10.0.0.2/32
  latest handshake: 6 seconds ago
  bandwidth: 377 B received, 520 B sent
peerA #
```

```
peerB # wg
interface: wg0
  public key: JkcUEgA9oqPHClu2l/j04dpBlxkipTG7sk
RoTBlFnH0=
  private key: sA6ljUB+0+nAmeAcU5iJ8xZrtWVkiia//
mkui9QO21E=
  listening port: 51820

peer: OjZc5ZDVoejDyBPX8ZZ5OwazzGcBy/nlfRAqKwaMok
U=
  endpoint: 192.168.1.1:51820
  allowed ips: 10.0.0.1/32
  latest handshake: 5 seconds ago
  bandwidth: 433 B received, 464 B sent
peerB #
```

Interface information is available

- Including traffic statistics

Demonstration—10/10

(**Aside:** As a software community, can we please do these sorts of videos more often to demonstrate new software?)

Cryptokey routing—1/2

- You define *which addresses* are valid for *which keys*.
- WireGuard checks that configuration when processing packets.
- Traffic accepted encrypted with key A has to be from one of key A’s addresses.

```
[Peer]
PublicKey = aGV5LCBnbyBhd2F5LCB0aGlzIGlzIGEgc2VjcmV0IQo=
AllowedIPs = 192.0.2.1/32, 198.51.100.0/24
```

Cryptokey routing—2/2

- You can set **networks** in `AllowedIPs`.
- This allows one peer to act as a VPN gateway for its configured peers:

```
[Peer]
PublicKey = bG9sIHlvdSBtdXN0IGJlIHZlcnkgYm9yZWQgICAgIAo=
AllowedIPs = 0.0.0.0/0
```

Network namespaces—1/7

- Linux has a feature called **network namespaces**.
- You can create (or move) separate interfaces, with their own addresses and routing tables, into separate **namespaces**.
- Processes can be set to run in a network namespace.
- I like to use systemd for this, but there are other ways.

Network namespaces—2/7

WireGuard has a cool property that works well with Linux network namespaces:

> WireGuard interfaces “remember” the namespace they were created in, via the original UDP socket, and continue passing traffic back through that endpoint, *even when moved to another namespace*.

Routing & Network Namespace Integration

Network namespaces—3/7

- *“Err…translation?”*
- Suppose you have two LibreWolf browser profiles on your home computer:
  - One, you need to run through a WireGuard VPN to your workplace (version control, work wiki…)
  - The other, you want to keep using on your home LAN as normal (home NAS, girlfriend’s calendar…)

Network namespaces—4/7

- Create a blank WireGuard interface.
- Create a new network namespace named “work”.
- Move the blank WireGuard interface into the “work” namespace.
- Add addresses, configuration, routes, DNS servers. etc, to the WireGuard interface.
- Use firejail or a similar tool to run one browser instance inside the “work” namespace.
- Now you can Alt-Tab between work browser and home browser…
- …or any other network program.

Network namespaces—5/7

```
$ cat /etc/network/interfaces.d/wg0
auto wg0
iface wg0 inet manual
        up ip netns add work
        up ip link add $IFACE type wireguard
        up ip link set $IFACE netns work
        up ip -n work address add 192.0.2.2/24 dev $IFACE
        up ip netns exec work wg setconf $IFACE /etc/wireguard/$IFACE.conf
        up ip -n work link set dev $IFACE up
        up ip -n work route add default via 192.0.2.1
        down ip netns del work

$ firejail --netns=work librewolf -P work --no-remote
```

Network namespaces—6/7

- **Warning:** The first time you get this working, you'll get mad you had to wait until 2022 to be able to do it.
- WireGuard is like that…

Network namespaces—7/7

- If you use Docker or similar container software, you're probably already getting ideas here.
- You can manufacture secure, cryptographically-authenticated network interfaces on your root namespace, and pass them in to your containers as their *only* contact with the outside world.
- This is left as an exercise for the viewer…

Questions?

WireGuard website

Email: tom@sanctum.geek.nz
Website: https://sanctum.geek.nz/
Fediverse: @tejr@mastodon.sdf.org