Implementing Routing Domains on an OpenBSD workstation for use with WireGuard

Josh Grosse
BSDCan 2024

Who is this presenter?

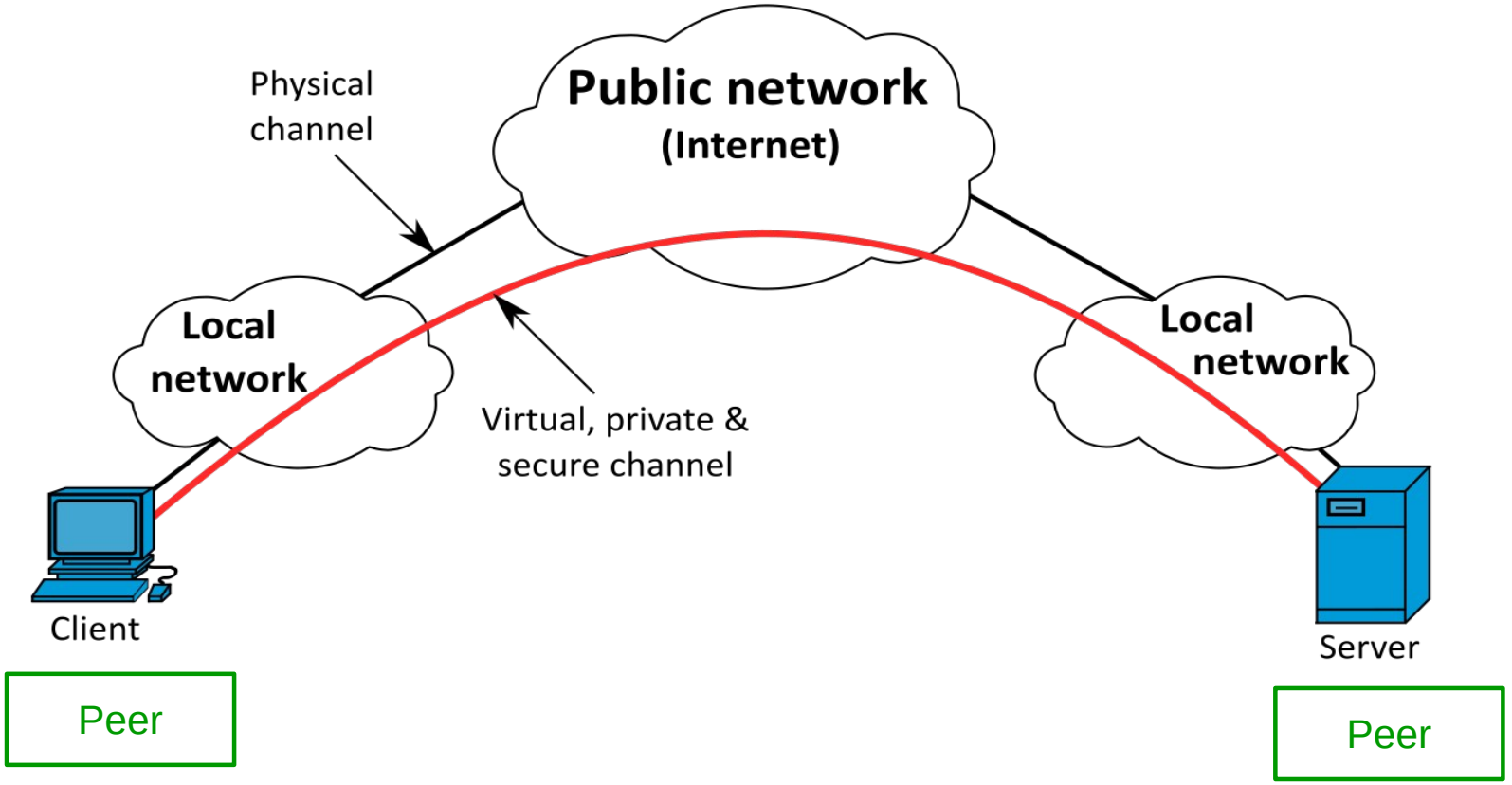
- OpenBSD user for more than 20 years
  - Port maintainer for a handful of third party packages
  - OpenBSD fan / hobbyist
  - Active participant at DaemonForums.org (user “jggimi”)
- IT professional since 1977
  - Applications programmer
  - Systems programmer (“sysadmin”)
  - Systems engineer
  - Manager (Product, Marketing, International Ops, M&A Contracts)

Agenda

- Background
  - VPNs in general
  - WireGuard in particular
- WireGuard Years 1-2: Routing by Priority
  - Automatic, worked well
  - All-or-nothing, on or off
- WireGuard Years 3-5: Routing by Domains
  - Technique recommended by Solène Rapenne (solene@)
  - Default: use the VPN
  - Optional: do not use the VPN

VPNs: Virtual Private Networks

"VPNs route private traffic over public networks."



VPN Implementations

Userland

- OpenVPN is a common example
- An application runs on the OS, acting as the transport facility between real and virtual networks
- The application may manage routing
- A tun(4) or similar pseudo-device may be used for communication

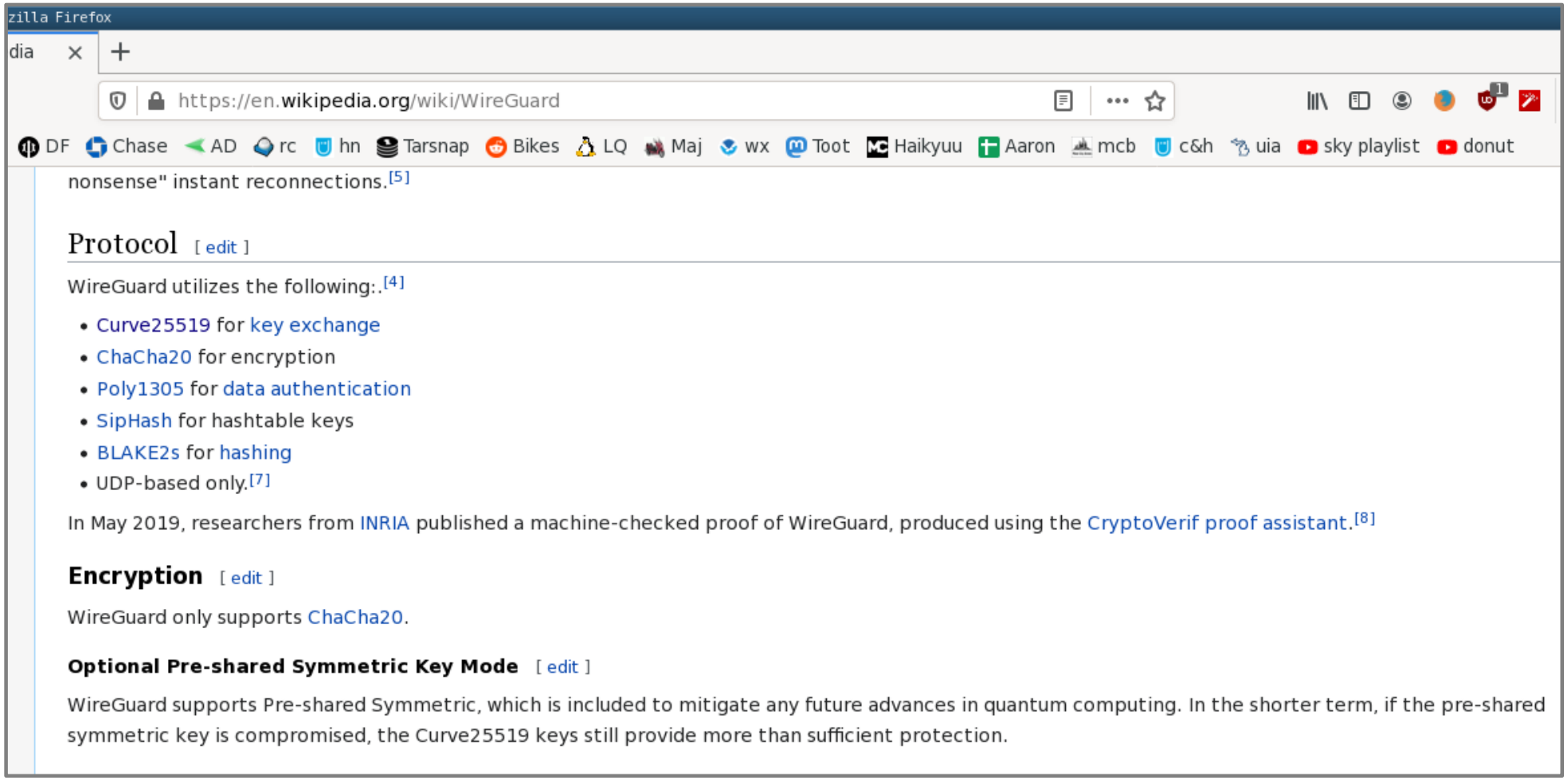
Kernel

- IPSec is a common example
- The kernel acts as the transport facility between the real and virtual networks
- The kernel manages routing

Why WireGuard?

- A relatively recent VPN technology (ca. 2016)
- Physical transport is via UDP
- Simple to provision and deploy
- Variety of OS implementations
- A kernel implementation on OpenBSD
  - Provisioned through the wg(4) pseudo-device driver
  - Familiar mechanism: there are 34 pseudo-device drivers on OpenBSD
    - *trunk(4), carp(4), vlan(4), bridge(4)...*
- Peer-to-Peer
  - Point-to-point, mesh, star (client / server)
  - IPv4 or IPv6 or a blended deployment
    - Either version may transport either

WireGuard uses well-regarded cryptographic primitives and protocols



Yay! (or, warning!) WireGuard uses simplified key management

- No certificates
  - No expirations
  - No renewals
  - No revocations – compromised keys must be changed manually
- All keys are 32 bytes long
    - Private keys are encoded in Base64 ASCII
    - Public* keys are derived automatically
    - Optional pre-shared keys for *"limited non-forward secret post-quantum resistance"* in key exchanges
- Many third-party admin tools are available
  - Geared for larger deployments, and never tested by me

** Operationally always treat WireGuard public keys as private. Identity hiding is a cryptographic requirement.*

Typical wg(4) provisioning

$ cat /etc/hostname.wg0*

```
wgkey <this private key>
wgpeer <that public* key> wgaip <ip block> wgaip <ip block>
wgpeer <that public* key> wgendpoint <ip> <port>
wgpeer <that public* key> wgpsk <pre-shared key>
inet 192.168.99.3/24
inet6 fd00::3/64
```

Note the private network addresses, RFC1918 and ULA

What's this machine's public* key?

 # ifconfig wg0 | grep pub

```
wgpubkey <this public key>
```

** Operationally always treat WireGuard public keys as private. Identity hiding is a cryptographic requirement.*

Agenda

- Background
  - VPNs in general
  - WireGuard in particular
- WireGuard Years 1-2: Routing by Priority
  - Automatic, worked well
  - All-or-nothing, on or off
- WireGuard Years 3-5: Routing by Domains
  - Technique recommended by Solène Rapenne (solene@)
  - Default: use the VPN
  - Optional: do not use the VPN

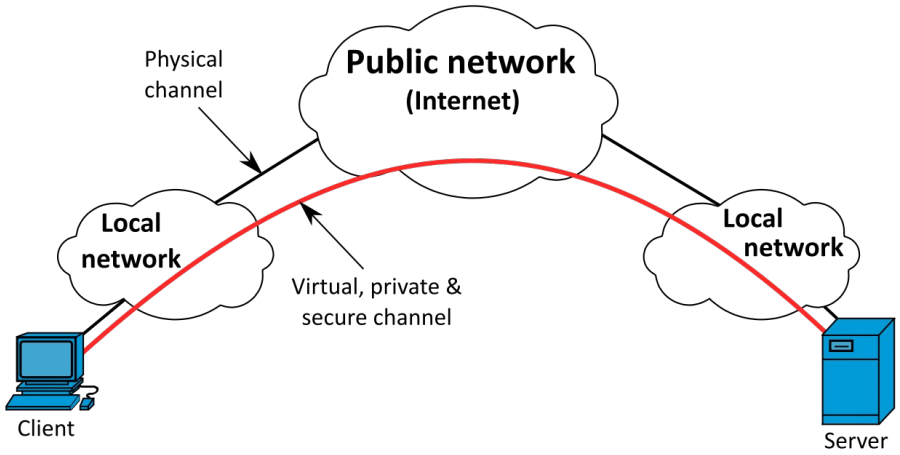
WireGuard Years 1-2: Routing by Priority

"VPNs route private traffic over public networks."

- Deployment decisions
  - Based upon my perceived requirements
    - What I thought I needed
    - How I thought it would function
  - Informed by limited experiences
    - Prior routing experience
      - I'd used *# route add*
      - I could describe "next hop" routing concepts
    - Prior VPN deployment experience
      - Several years of IPSec point-to-point
      - A week of OpenVPN "testing" ca. 2003
      - End user of a dozen corporate VPN userland clients

My perceived needs

“VPNs route private traffic over public networks.”



Requirements:

- A default route over the VPN
- A backup default route when the VPN was not available – by intent or by accident
- A specific route to the remote endpoint through the public network – to operate the VPN

Routing by priority

- Whenever two routes are both operational, the route with the highest priority gets used:
  - 12: Low priority routes: default routes on the physical network.
  - 7: Medium priority routes: default routes using the VPN.
  - 2: High priority route: physical route to the remote server.
- Simple implementation:
  - Include -priority <n> on # route add commands
  - Can be provisioned either in hostname.if(5) or in rc.local(8)
  - Turn VPN off with # ifconfig wg0 down
  - Turn VPN on with # ifconfig wg0 up

Provisioning priority routing

- The lower the -priority number ... the higher the priority
- Auto-configured default routes are currently set to -priority 8

  $ cat /etc/rc.local *(circa 2021)*

```
# [this peer 192.168.99.3 / ff0d::3] – [gateway peer 192.168.99.1 / ff0d::1]

route add -priority 7 default 192.168.99.1
route add -priority 7 -inet6 default fd00::1
route add -priority 2 <real peer endpoint> <real next hop>
```

Agenda

- Background
  - VPNs in general
  - WireGuard in particular
- WireGuard Years 1-2: Routing by Priority
  - Automatic, worked well
  - All-or-nothing, on or off
- WireGuard Years 3-5: Routing by Domains
  - Technique recommended by Solène Rapenne (solene@)
  - Default: use the VPN
  - Optional: do not use the VPN

Solène Rapenne’s Guide:

“Full WireGuard setup with OpenBSD”

https://dataswamp.org/~solene/2021-10-09-openbsd-wireguard-exit.html

“Some have seen further by standing upon the shoulders of giants.
I was able to see further, because solene@ opened my eyes.”

WireGuard Years 3-5: Routing by Domains

“VPNs route private traffic over public networks.”

- Why switch?
  - I’d read solene@’s guide, and foresaw benefits
    - Application granularity
    - Know when the VPN was down
- How?
  - The VPN uses the default routing domain – rdomain 0 / rtable 0
    - All processes use rdomain 0 by default
  - The physical network uses rdomain 1 / rtable 1
    - Anything that need the real network must be run from rdomain 1
  - Process-level assurance of the network in use
  - The parent process and all of its children are fixed to a single routing domain
  - Switching domains means restarting the application

Route decision is made at process start

$ firefox

$ novpn chrome

$ cat ~/bin/novpn

```
#!/bin/sh
route -T 1 exec $@
```

- Shell script rather than an alias – for simplified WM provisioning

Provisioning rdomain 1 / rtable 1 (part 1)

- Put the egress NIC in rdomain 1
  (my egress NIC happens to be a trunk(4) pseudo-device)

  $ cat /etc/hostname.trunk0

```
rdomain 1
trunkproto failover
trunkport em0
trunkport iwm0
inet autoconf
inet6 autoconf
```

  - The *trunkport* devices are also in rdomain 1

    e.g.: *$ cat /etc/hostname.em0*

```
rdomain 1
up
```

Provisioning rdomain 1 / rtable 1 (part B)

- Loopback for rdomain 1 / rtable 1

  $ cat /etc/hostname.lo1

```
rdomain 1
inet 127.0.0.1/8
inet6 ::1/128
```

- Add a *wgrtable* directive

  # cat /etc/hostname.wg0

```
wgkey …
… wgrtable 1 ...
inet 192.168.99.3/24
inet6 fd00::3/64
!route add default 192.168.99.1
!route add -inet6 default fd00::1
```

From ifconfig(8) for “wgrtable <rtable>”

“Exchange traffic with peers under the routing table <rtable>, instead of the default.... [It] needn't be the routing domain to which the interface is attached, in which the interface's tunneled traffic appears.”

The default routing table: rtable 0

$ netstat -nrf inet

```
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use   Mtu  Prio Iface
default            192.168.99.1       UGS         0      312     -     8 wg0
127/8              127.0.0.1          UGRS        0        0 32768     8 lo0
127.0.0.1          127.0.0.1          UHhl        3      466 32768     1 lo0
192.168.99/24      192.168.99.3       UCn         1        0     -     4 wg0
192.168.99.1       link#0             UHch        1        3     -     3 wg0
192.168.99.3       wg0                UHl         0      103     -     1 wg0
192.168.99.255     192.168.99.3       UHb         0        0     -     1 wg0
```

The physical routing table: rtable 1

$ netstat -T 1 -nrf inet

```
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use   Mtu  Prio Iface
default            10.0.1.1           UGS         7    14590     -     8 trunk0
10.0.1/24          10.0.1.130         UCn         2      836     -     4 trunk0
10.0.1.1           00:00:5e:00:01:01  UHLch       1      743     -     3 trunk0
10.0.1.130         50:7b:9d:3b:16:ca  UHLl        0    44615     -     1 trunk0
10.0.1.254         00:0d:b9:2f:9a:7c  UHLc        1      588     -     3 trunk0
10.0.1.255         10.0.1.130         UHb         0        0     -     1 trunk0
100.64.3.2/31      100.64.3.2         UCn         0        0     -     4 vport0
100.64.3.2         fe:e1:ba:d0:16:86  UHLl        0        0     -     1 vport0
127.0.0.1          127.0.0.1          UHl         1     7592 32768     1 lo1
```

Guest VMs use rdomain 1 / rtable 1

 # grep domain /etc/pf.conf

```
# nat guest VMs to the appropriate egress by routing domain
pass out on rdomain 1 from 100.64.0.0/10 to any rtable 1 nat-to ($ext)
pass in on rdomain 1 from 100.64.0.0/10 to (self) rtable 1
# redirect guest domain requests to unwind(8)
pass in proto { tcp udp } from 100.64.0.0/10 to any port domain rdr-to localhost
```

I use two instances of unwind(8)

*$ ls -l /etc/rc.d/unwind**

```
-r-xr-xr-x  1 root  wheel  256 May 15 14:41 /etc/rc.d/unwind
lrwxr-xr-x  1 root  wheel    6 Sep 29  2023 /etc/rc.d/unwind1 -> unwind
```

$ grep unwind /etc/rc.conf.local

```
pkg_scripts=unwind1 …
unwind1_flags=-s /dev/unwind1.sock
unwind1_rtable=1
unwind_flags=
```

Final thoughts on WireGuard

- You are the architect of your own VPN topology.
  - My VPN is both “mesh” and “star”
    - Mesh: communication between servers
    - Star: workstations / mobile → Internet gateway
    - 6 platforms: OpenBSD, Android, Windows
- The allowed IP blocks: “wgaip”
  - Filter packets: by source IP address on incoming tunneled packets
  - At least one address or block is required
  - Can be ::0/0 or 0.0.0.0/0 (“do not filter packets”)

Questions?

Permission to use image from https://reverse.mortgage/