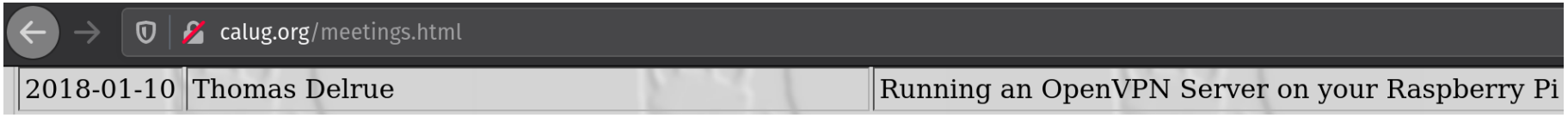
WireGuard

Tunneling with simplicity and usability

Intro & Disclaimers

- Hi
- I’m not an expert (yet)
  - I’m not a cryptologist either
- YMMV
- I gave an OpenVPN presentation here before
  - …yeeaaaah… we all grow…?



Executive Summary

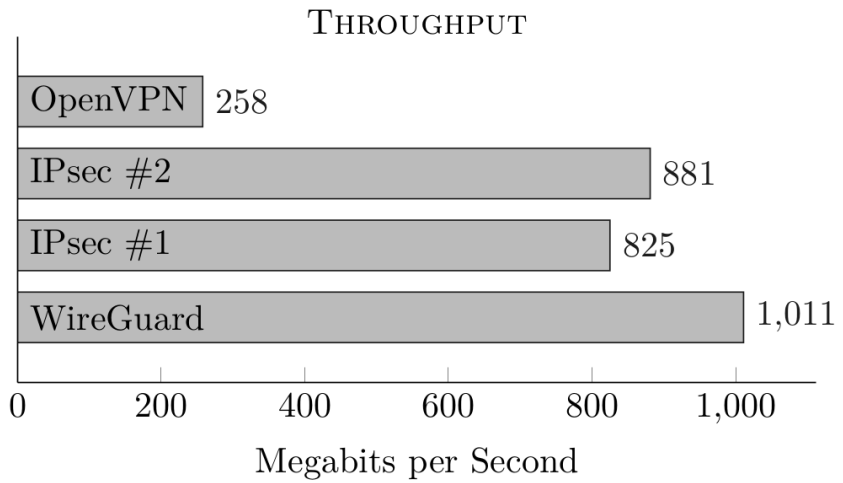
- WireGuard is a lightweight VPN solution
  - Aims to replace IPsec & OpenVPN
  - Practical, usable and simple
    - i.e. not some academic research mumbo-jumbo
  - Uses tunnels-as-in-SSH, not TunneLS-as-in-TLS
- It’s Open Source
- It’s secure (formal verification of the protocol)
  - It’s small, which helps (4000 LOC)
- It’ll be in Linux Kernel 5.6

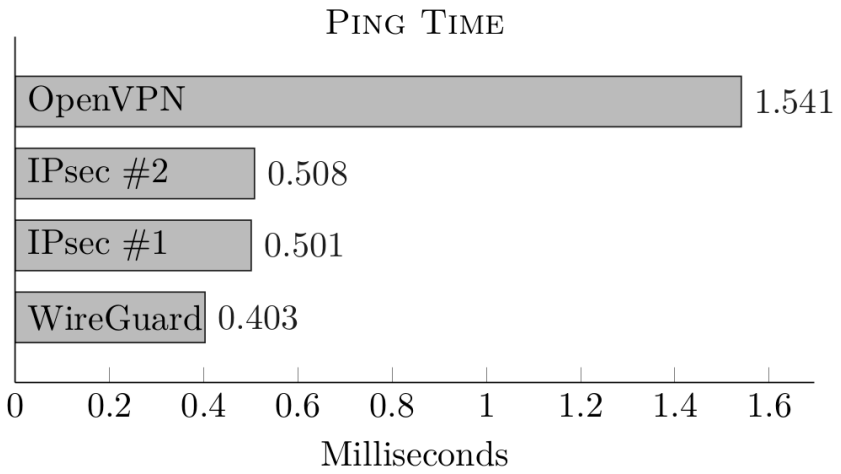
OpenVPN & IPsec

- Both are cool
  - If you know what you’re doing
  - And you want to deal with them
- OpenVPN
  - Relatively high level of complexity
  - Management is finicky
  - Uses OpenSSL, a large & complex code-base
    - Not intended as a judgment, just an observation
- IPsec
  - High level of complexity
- Both OpenVPN & IPsec are easy to mess up
  - And then cause massive headaches

WireGuard vs OpenVPN/IPsec

Protocol	Configuration
WireGuard	256-bit ChaCha20, 128-bit Poly1305
IPsec #1	256-bit ChaCha20, 128-bit Poly1305
IPsec #2	256-bit AES, 128-bit GCM
OpenVPN	256-bit AES, HMAC-SHA2-256, UDP mode





- Numbers from the white-paper
- Bottlenecks
  - OpenVPN & IPsec tests showed 100% CPU utilization
  - WireGuard did not utilize CPU at 100%
  - Suggests that WireGuard saturated the link, i.e. the network bandwidth was the bottleneck, not the CPU
- OpenVPN is user-space, IPsec & Wireguard are not, which also explains the huge discrepancy shown by OpenVPN

Theory

- Layer 3/Network layer
  - Packet Routing and Forwarding
- Shows up as a Virtual Network Interface
  - It’s just a device
  - e.g. /dev/wg0
- Device can be managed using ip or ifconfig
  - add type wireguard, delete, up, down, routes
- Firewall rules are simple
  - e.g. iptables -A *chain* -i wg0 -j *rule*
- Kernel-space
  - Currently as DKMS/Dynamic Kernel Module Support
  - Performance

Theory: Crypto and Security

- Encryption “at the device”
  - You just see the device
  - ClearText → *device* → CipherText → *device* → ClearText
- Perfect Forward Secrecy
  - Compromised session key != compromised private key
- Key Exchange similar to OpenSSH
  - Key Size is 32bytes (256bits/44 chars in base64)
    - EC Crypto (I’m not a cryptologist, I’m just parroting)
- Authenticated packets only: silence is golden
  - Everything else is dropped at the device
  - Not-dropped/routed therefore authenticated & good
- Stateless
  - No state to attack or trick
- Routing based on crypto keys

Theory: CryptoKey Routing

- Routing happens based on peer's public key
- Every interface has its own:
  - Public-private key pair
  - UDP port
  - Routing table of pubkeys → allowed source IP
- When sending packets
  - Destination determines session key for encryption
- When receiving packets
  - if source IP does not match the decryption key (i.e. it cannot be authenticated), then the packet is rejected
  - Thus, if a packet is not rejected, it is from an authenticated source
  - This addresses a set capabilities of IPsec

Theory: Flow – Sending

- ClearText packet enters /dev/wg0
- Packet is encrypted (using ChaCha20Poly1305) for the destination
  - If no peer exists, then -ENOKEY (i.e. no route to host) is returned
- Encrypted payload encapsulated with headers into a packet
- Encrypted UDP packet is sent to destination

Theory: Flow - Receiving

- UDP packet received on WireGuard port
  - This is not the same as the device
- Headers inform WireGuard which peer (i.e. public/session key) to use
  - Validate and drop if validation fails
- Updates the endpoint of the Peer
  - To allow for roaming and UDP is session-less
- Decrypt packet payload
- If the packet needs forwarding, it is forwarded
- If the packet is for this host, insert into device

Practicum: Install & Prepare

- Add PPA and install WireGuard (client & server)

```
$> add-apt-repository ppa:wireguard/wireguard
$> apt update
$> apt install wireguard
```

- Set up IP forwarding on server: /etc/sysctl.conf

```
net.ipv4.conf.all.forwarding=1
net.ipv4.ip_forwarding=1
net.ipv6.conf.all.forwarding=1
net.ipv6.conf.default.forwarding=1
```

Practicum: Generate Keys

- You need a private/public key (client & server)
  - Generate Keys

```
$> wg genkey > privatekey
$> wg pubkey < privatekey > publickey
```

  - In 1 go

```
$> wg genkey | tee privatekey | wg pubkey > publickey
```

  - Security:

```
$> chmod -R 600 *key # or do the commands above with umask 077
```

Practicum: Configure Server

- /etc/wireguard/wg0.conf (set permissions!!)

```
[Interface]
Address = SERVER_CIDR # e.g. 10.0.0.1/32
SaveConfig = false # Don't overwrite what we do here on service stop
PostUp = iptables -I INPUT 1 -i %i -j ACCEPT; iptables -A FORWARD -i %i -j
ACCEPT; iptables -A FORWARD -o %i -j ACCEPT; iptables -t nat -A POSTROUTING
-o eth0 -j MASQUERADE
PostDown = iptables -D INPUT -i %i -j ACCEPT; iptables -D FORWARD -i %i -j
ACCEPT; iptables -D FORWARD -o %i -j ACCEPT; iptables -t nat -D POSTROUTING
-o eth0 -j MASQUERADE
ListenPort = 23456 # or whatever
PrivateKey = CONTENT_OF_PRIVATEKEY_FILE
# PublicKey = CONTENT_OF_PUBLICKEY_FILE (to have a record)

[Peer]
PublicKey =  PEER_00_PUBLICKEY
AllowedIPs = PEER_00_ALLOWED_IP_CIDRS

[Peer]
PublicKey =  PEER_01_PUBLICKEY
AllowedIPs = PEER_01_ALLOWED_IP_CIDRS
```

Practicum: Start Server

- SystemD

```
$> sudo systemctl start wg-quick@wg0.service
$> sudo systemctl enable wg-quick@wg0.service
```

- Inspect WireGuard

```
$> sudo wg
interface: DEVICE_NAME
  public key: PUBLICKEY_HERE
  private key: (hidden)
  listening port: MY_PORT_HERE

peer: PUBLICKEY_OF_OTHER_SIDE
  endpoint: MY_IP:443
  allowed ips: CIDRS_HERE
  latest handshake: 3 seconds ago
  transfer: 46.33 KiB received, 41.50 KiB sent
  persistent keepalive: every 25 seconds

[...more peers here...]
```

Practicum: Configure Client

- /etc/wireguard/wg0.conf

```
[Interface]
Address = YOUR_VPN_IP_IE_HOW_YOU_LL_BE_KNOWN
PrivateKey = YOUR_PRIVATEKEY
ListenPort = YOUR_INBOUND_PORT

[Peer]
PublicKey = YOUR_SERVERS_PUBLICKEY
Endpoint = YOUR_SERVER:YOUR_PORT
AllowedIPs = 0.0.0.0/0, ::/0    # what to route through here (here everything)
PersistentKeepalive = 25
```

- Bring the interface up

```
$> sudo wg-quick up wg0
```

- See also WireGuard's quickstart for what wg-quick actually does behind the scenes

VPN script

- I store my configurations in ~/.wireguard/*.conf
- Wrote a vpn script to manage my VPNs

```
$> vpn help
usage: vpn [list, status, {up|down} <name>]
  list          List all VPNs
  status        Shows the vpn status (equivalent to calling the script
without any arguments)
  up <name>     Bring up the VPN named <name>
  down <name>   Take down the VPN named <name>
```

VPN script: list

- All VPNs I have configured
- Lists ${HOME}/.wireguard/*.conf

```
$> vpn list
Available VPNs (from ${HOME}/.wireguard):
=========================================
  - attached_home
  - ${WORK}
  - home
```

- Configurations
  - home: pretend I’m home

    ```
    AllowedIPs = 0.0.0.0/0, ::/0
    ```

  - attached_home: get me access to my home resources but don’t route it ALL through there
    - `AllowedIPs = ${WG_SERVER_INTERFACE_24CIDR}, ${HOME_NETWORK_24CIDR}`
  - ${WORK}: route it all through work

VPN script: up/down

- Up: brings up the named VPN
  - As in ${HOME}/.wireguard/<name>.conf

```
$> vpn up home
[#] ip link add home type wireguard
[#] wg setconf home /dev/fd/63
[#] ip -4 address add ${WG_SERVER_INTERFACE_24CIDR} dev home
[#] ip link set mtu 1420 up dev home
[#] wg set home fwmark 51820
[#] ip -6 route add ::/0 dev home table 51820
[#] ip -6 rule add not fwmark 51820 table 51820
[#] ip -6 rule add table main suppress_prefixlength 0
[#] ip -4 route add 0.0.0.0/0 dev home table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] ${HOME}/.wireguard/home.script.bash PostUp
```

- Down: takes down the named VPN

```
[#] ${HOME}/.wireguard/home.script.bash PreDown
[#] ip -4 rule delete table 51820
[#] ip -4 rule delete table main suppress_prefixlength 0
[#] ip -6 rule delete table 51820
[#] ip -6 rule delete table main suppress_prefixlength 0
[#] ip link delete dev home
```

VPN script: status

- Shows the current status
- When no VPN active

```
VPN Status
==========

  Routes
  ------
  default     h.i.j.1 # my gateway
  a.b.0.0/16  wlan0
  e.f.g.0/24  virbr0
  h.i.j.0/24  wlan0
```

VPN script: status

- When VPN active (and routing everything)

```
VPN Status
==========
interface: home
  public key: MY_PUBLICKEY
  private key: (hidden)
  listening port: MY_PORT
  fwmark: 0xca6c

peer: MY_HOME_PUBLICKEY
  endpoint: MY_HOME_ENDPOINT
  allowed ips: 0.0.0.0/0, ::/0
  latest handshake: 3 seconds ago
  transfer: 412 B received, 5.43 KiB sent
  persistent keepalive: every 25 seconds

  Routes
  ------
  default     e.f.g.1 # my gateway
  a.b.0.0/16  wlan0
  e.f.g.0/24  wlan0
```

VPN script: status

- When VPN active (only routing 10.0.{1,2}.0/24)
  - Everything else bypasses WireGuard

```
VPN Status
==========
interface: attached_home
  public key: MY_PUBLICKEY
  private key: (hidden)
  listening port: MY_PORT
  fwmark: 0x1  # no fwmark?

peer: MY_HOME_PUBLICKEY
  endpoint: MY_HOME_ENDPOINT
  allowed ips: 10.0.1.0/24, 10.0.2.0/24
  latest handshake: 3 seconds ago
  transfer: 412 B received, 5.43 KiB sent
  persistent keepalive: every 25 seconds

  Routes
  ------
  default      e.f.g.1 # my gateway
  a.b.0.0/16   wlan0
  e.f.g.0/24   wlan0
  10.0.2.0/24 attached_home
  10.0.1.0/24 attached_home
```

VPN script: code

```
#!/usr/bin/env bash

VPN_SRC="${HOME}/.wireguard"
INDENT="  ";

function usage()
{
    echo "usage: vpn [list, status, {up|down} <name>]";
    echo "${INDENT}list          List all VPNs";
    echo "${INDENT}status        Shows the vpn status (equivalent to calling the script without any arguments)";
    echo "${INDENT}up <name>     Bring up the VPN named <name>";
    echo "${INDENT}down <name>   Take down the VPN named <name>";
}

function show_status()
{
    echo "VPN Status";
    echo "==========";
    ${WG};

    echo "";
    echo "${INDENT}Routes";
    echo "${INDENT}------";
    ip route | cut --delim " " --fields 1,3 | column -t -s' ' | sed "s/^/${INDENT}/g";
    echo "";
}

function list_vpns()
{
    echo "Available VPNs (from ${VPN_SRC}):";
    echo "================================================";
    pushd "${VPN_SRC}" > /dev/null;
    ls -1 *.conf | sed "s/.conf$//g" | sort | sed "s/^/${INDENT}- /g";
    popd > /dev/null;
}

function do_vpn_action()
{
    CMD="${1}";
    TARGET="${2}";
    ${WGQUICK} "${CMD}" "${VPN_SRC}/${TARGET}.conf";
}
```

```
#make sure we have our environment straight
WGQUICK="$(which wg-quick)";
if [ "${WGQUICK}" == "" ]
then
    echo "ERROR: wg-quick (and thus Wireguard) is not installed.";
    WGQUICK="echo FAKE: \"wg-quick\"";
else
    WGQUICK="sudo ${WGQUICK}";
fi;

WG="$(which wg)";
if [ "${WG}" == "" ]
then
    echo "ERROR: wg (and thus Wireguard) is not installed.";
    WG="echo FAKE: \"wg\"";
else
    WG="sudo ${WG}";
fi;

case "${1}" in
    "list")
        list_vpns;
        ;;
    "" | "status")
        show_status;
        ;;
    "up" | "down")
        if [ "${2}" == "" ]
        then
            usage;
        else
            do_vpn_action "${1}" "${2}";
        fi;
        ;;
    *)
        usage;
        ;;
esac;
```

Configuration Tricks

- [Interface]: PreUp/PostUp/PreDown/PostDown
  - Executes the specified executable
  - Examples:
    - PostUp: mount something, ping something
    - PreDown: unmount the thing you mounted
    - PostDown: modify firewall rules
  - e.g. in my [attached_]home.conf:

```
[Interface]
# ...other stuff...
PostUp = /home/someone/.wireguard/home.script.bash PostUp
PreDown = /home/someone/.wireguard/home.script.bash PreDown
```

  - Does not need to be the same script
- DNS
  - DNS Server(s) to use

Demo yourself

- Install WireGuard first
- Route one host only (192.168.4.1)

```
$> wget https://git.zx2c4.com/wireguard-tools/plain/contrib/ncat-client-
server/client.sh
$> sudo client.sh          # creates /dev/wg0
$> curl 192.168.4.1
$> sudo ip link delete wg0 # when done
```

- Route *all* your traffic (0.0.0.0/0)

```
$> curl icanhazip.com
$> sudo client.sh default-route # creates /dev/wg0
$> curl icanhazip.com      # should be different from the first one
$> sudo ip link delete wg0 # when done
```

Services offering WireGuard

- IVPN (beta)
- Mullvad (beta)
- You

Links

- https://www.wireguard.com/
- https://www.wireguard.com/quickstart/
- https://www.wireguard.com/papers/wireguard.pdf
  - Highly recommended read!
- https://en.wikipedia.org/wiki/Wireguard
- https://hal.inria.fr/hal-02100345v2/document
  - Formal cryptographic proof of the *protocol*
  - June 2019