Setup Guide OpenVPN Master branch

Version 0.7
Latest guide: OpenVPN setup guide Master branch

Table of Contents

Introduction....1
Installation....1
Prerequisites....1
Prerequistes for an OpenVPN Server....1
Check all involved subnets....2
Check for Public IP address....2
Test from outside....2
Using Config File....2
Setup Interface....4
Setup Firewall....8
OpenVPN Client....8
OpenVPN Server....9
Status....9
Management interface....9
Problems....10
Addendum 1 Script to convert config to uci....11

Introduction

On master branch *luci-app-openvpn* is replaced by *luci-proto-openvpn*.
There is no longer a separate entry in the GUI for OpenVPN but it has become a protocol so it is integrated in the interface.
The whole interface and inner working are redesigned so this guide is only applicable for **Master branch** as 25.12 and earlier branches use the old style luci-app-openvpn

This guide and also the OpenVPN implementation is very much a Work in Progress and subject to change.

Installation

If you do not have OpenVPN installed at all then install by first by updating APK and then installing *openvpn-openssl* and *luci-proto-openvpn*

Prerequisites

A fully functioning router with the correct time.

Prerequistes for an OpenVPN Server

Skip this paragraph for setting up an **OpenVPN Client** and **continue** to **Using Config File**

Check all involved subnets

For setting up an OpenVPN **tun** server all three involved subnets have to be different and non overlapping.
So the routers subnet, the OpenVPN subnet and the Clients subnet all have to be different.
As you often cannot choose the subnet of the OpenVPN client, it is best to **avoid** using frequently used subnets for your router e.g. *192.168.1.1/24* or *192.168.0.1/24* but if you cannot easily change the routers subnet then leave it and hope for the best.
For the OpenVPN subnet *172.23.23.1/24* is chosen in this example which is not often used.

Check for Public IP address

To be able to connect from outside, your router must have a public IP address (either IPv4 and/or IPv6).
Check if your router has a proper Public IPv4 address with (from command line):
ifstatus wan | grep address

A public IP address does **not** start with 192.168.X.X, 10.X.X.X, 172.16-31.X.X or a CGNAT address (IP addresses from 100.64.0.0 to 100.127.255.255)

Another way to test is to compare the routers IP address (*ifstatus wan | grep address)* with the address from *ipleak.net* it should show the same address for a Public IPv4 address. If it is not the same you might not have a Public IPv4 address and the router is not reachable with IPv4.

If your router is behind another router then:
- Check if that router has a Public IP address.
- Check if and how you can Port Forward from that router to your router which is going to run the OpenVPN Server.

If you are behind CGNAT, so do not have a public IPv4 address and do not have a public IPv6 address (check with: *ifstatus wan6*) or using IPv6 is not applicable then you have to involve a commercial third party to get a public IP address.
This can be a VPN provider which supports port forwarding (e.g. AirVPN, ProtonVPN), or you can rent a Virtual Private Server (I have an Oracle VPS which can be had for free, see at the bottom of this guide), or use things like Netbird, Zerotier, Cloudflared, Tailscale, ngrok, pinggy, or tunnelmole and there are more, I have setup Netbird on several OpenWRT and Windows and Linux clients and it works well, see my notes about setting up Netbird on OpenWRT and the Netbird support thread.

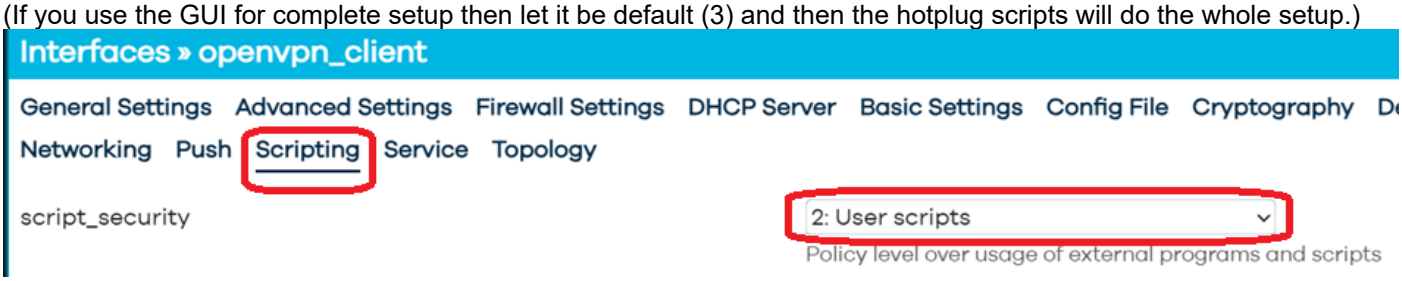
If your Public IP address is non static, e.g. it can change, then look into using DDNS.

Test from outside

Proper testing can only be done from outside e.g. with your phone or laptop on cellular data or from a friends/neighbors internet.

Using Config File

When using your own config file to setup an OpenVPN Client set script-security 2
(If you use the GUI for complete setup then let it be default (3) and then the hotplug scripts will do the whole setup.)



This will allow you to setup the VPN tunnel to you own liking

/var/run/openvpn.openvpn_client.conf:

```
cd /etc/openvpn/openvpn_client
status /var/run/openvpn.openvpn_client.status
syslog openvpn_openvpn_client
tmp-dir /tmp
script-security 2
dev tun1
dev-type tun
mtu-disc no
config '/etc/openvpn/openvpn_client/openvpn_client_config.cfg'
```

If you leave the script security at its default which is 3 then the hotplug script will set everything up for you. This reads the variables from the OpenVPN environment so it will setup a tunnel with a config file but no routing is setup as *route-noexec* is added to the config file.

For an OpenVPN Server you could try and leave the script-security 3 as that will show traffic on the interface but you can not set routes via the script but for a simple setup this could work.

/var/run/openvpn.openvpn_client.conf with script-security 3:

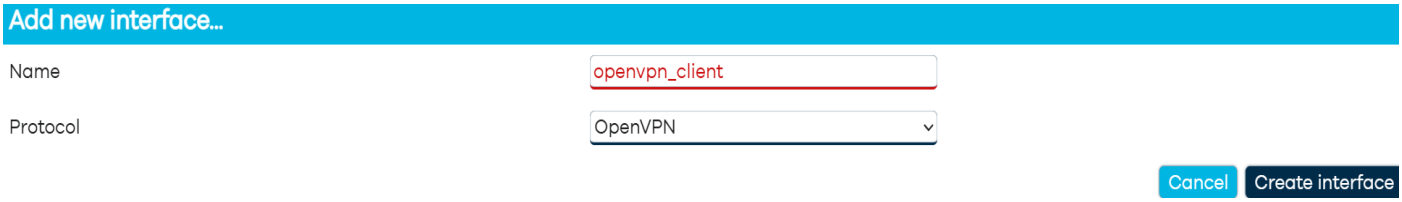
```
cd /etc/openvpn/openvpn_client
status /var/run/openvpn.openvpn_client.status
syslog openvpn_openvpn_client
tmp-dir /tmp
script-security 3
dev tun1
dev-type tun
mtu-disc no
config '/etc/openvpn/openvpn_client/openvpn_client_config.cfg'
setenv INTERFACE openvpn_client
script-security 3
setenv IPV6 '1'
ifconfig-noexec
route-noexec
up '/usr/libexec/openvpn-hotplug'
down '/usr/libexec/openvpn-hotplug'
route-up '/usr/libexec/openvpn-hotplug'
route-pre-down '/usr/libexec/openvpn-hotplug'
tls-crypt-v2-verify '/usr/libexec/openvpn-hotplug'
```

Even if you use the config file it will work to some extent when setting script security to 3 as it still reads from the config file. So you can use a hybrid setup.

When using the GUI to setup, you can use the config file to add options not in the GUI e.g. *pull-filter ignore "redirect-gateway ipv6"* or *remote-random* etc.

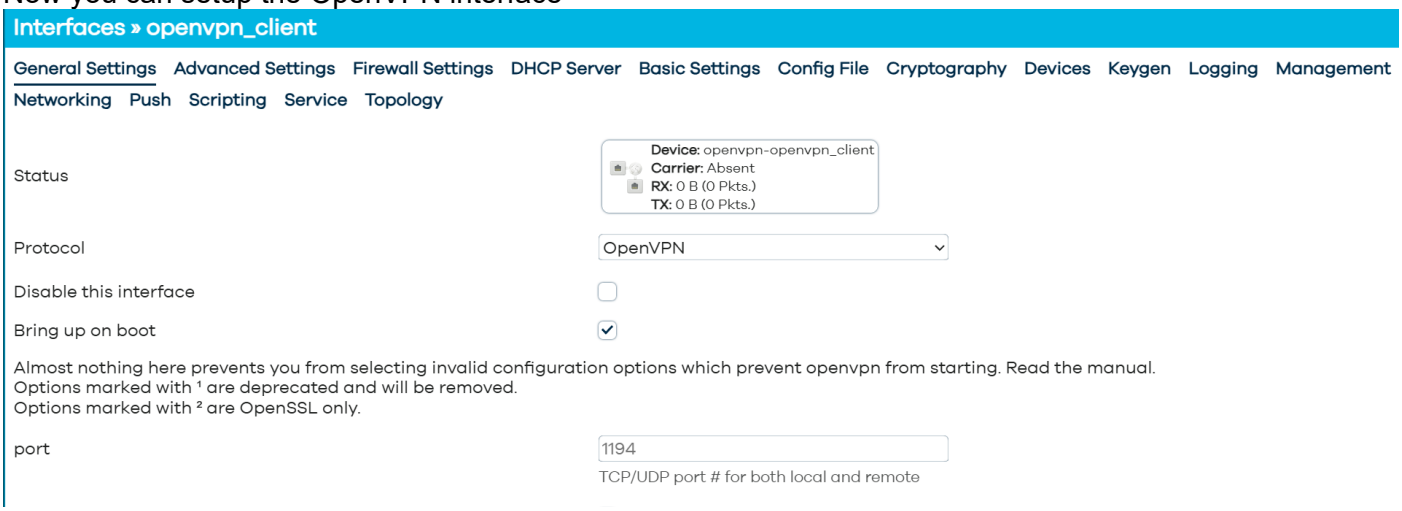
Setup Interface

To setup OpenVPN (whether client or server)
Network > Interfaces > Add new interface
Name: choose a name not more than 15 characters, I use *openvpn_client* in this example
Protocol: OpenVPN.

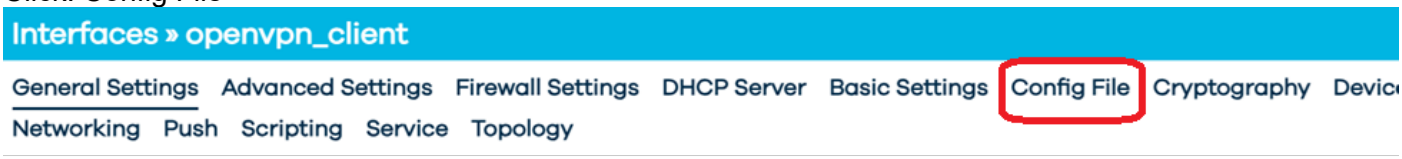


Click:
Create interface

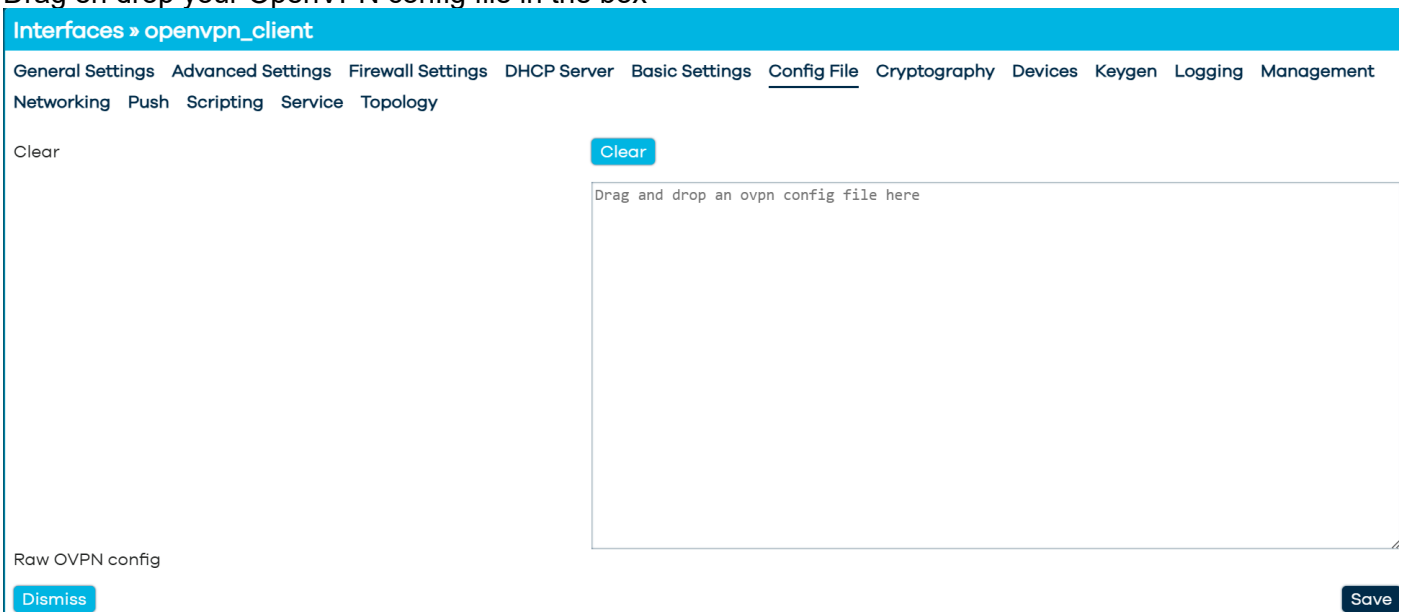
Now you can setup the OpenVPN interface



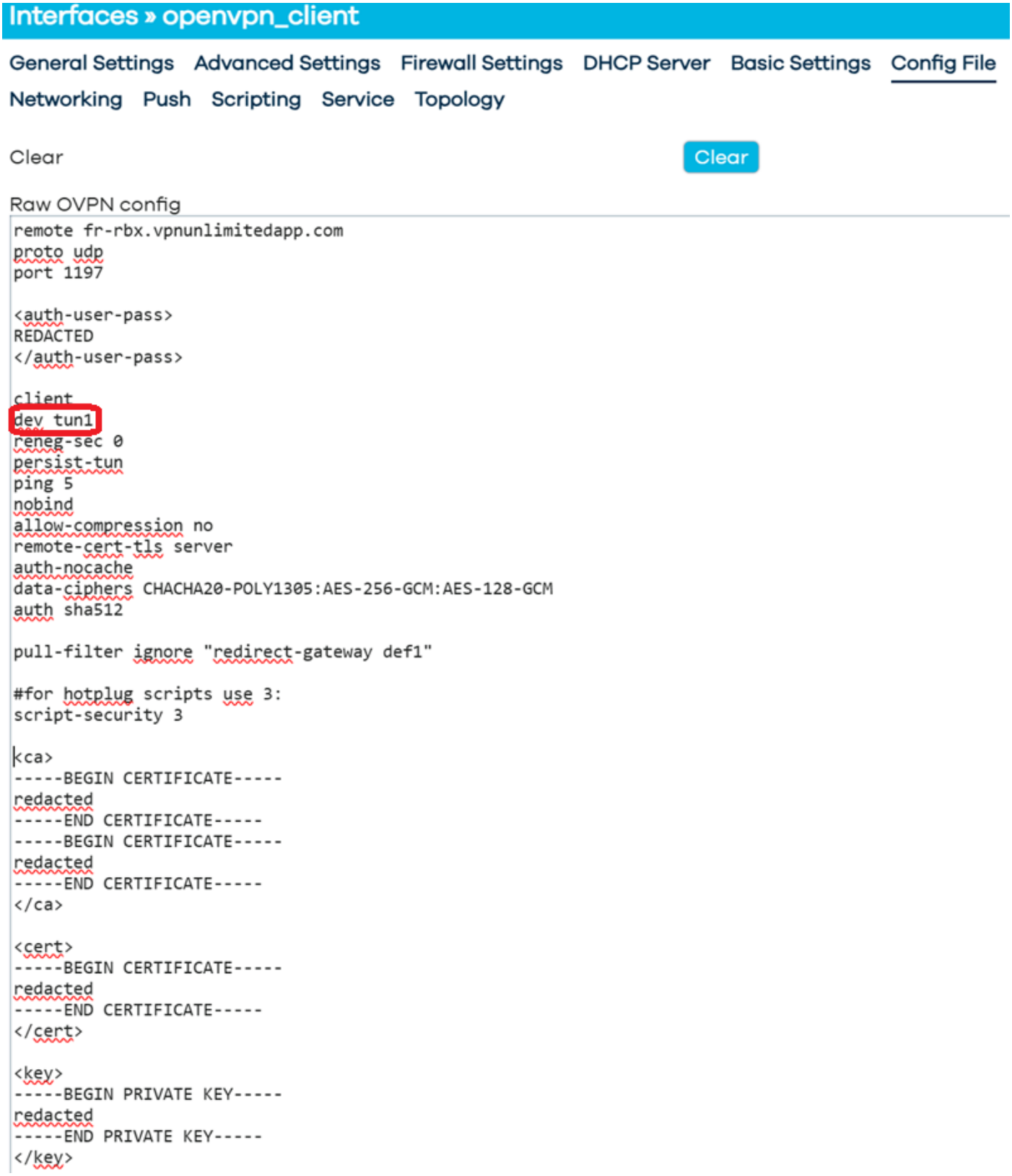
Import your config file into the interface.
Click: Config File



Drag en drop your OpenVPN config file in the box



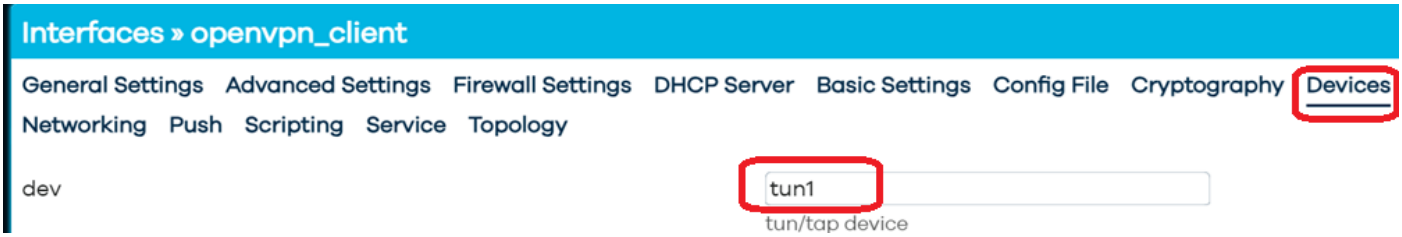
Config files can contain all necessary info *inline* including keys certificates and username and password as shown below



Often a config file just has *dev tun* so without a number, in this case OpenVPN itself will use the first available number which is **0** so the device will be tun0
I think this is bad practice and encourage you to choose a number for the device.
I use *tun1* for my client and *tun2* for my server.

As is chosen to setup the tunnel with a config file with script-security 2 there is no information about the used *l3_device* (the *tun1* device in this example) to make sure applications which use the interface can

extract the l3_device (e.g. tun1) from the config (e.g. for Policy Based Routing) also add the device under Devices > dev:



this way PBR wil now the device used by the openvpn_client interface

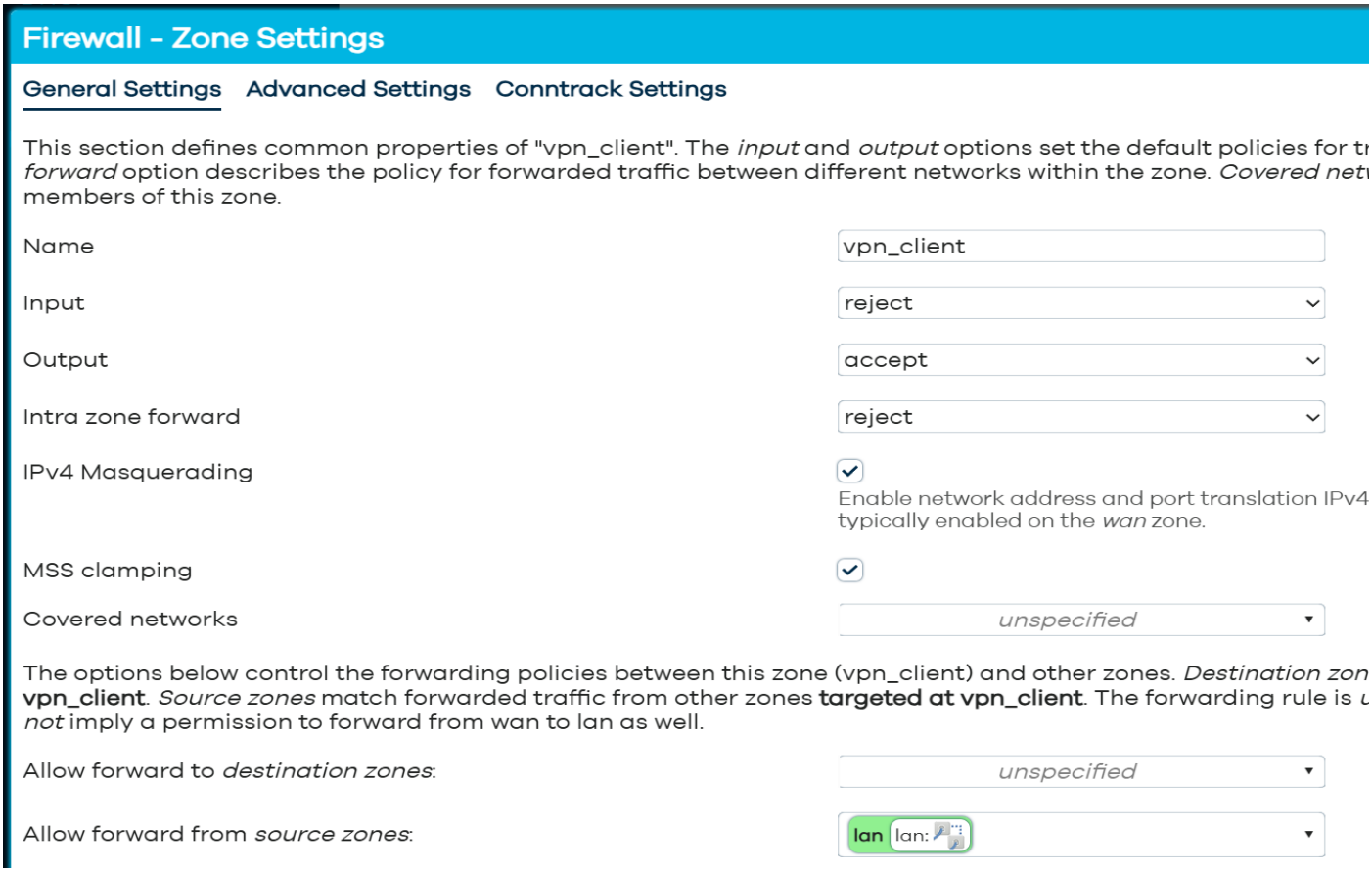
/etc/config/network/

```
config interface 'openvpn_client'
        option proto 'openvpn'
        option dev_type 'tun'
        option mtu_disc 'no'
        option multipath 'off'
        option config '/etc/openvpn/openvpn_client/openvpn_client_config.cfg'
        option script_security '2'
        option dev 'tun1'
```

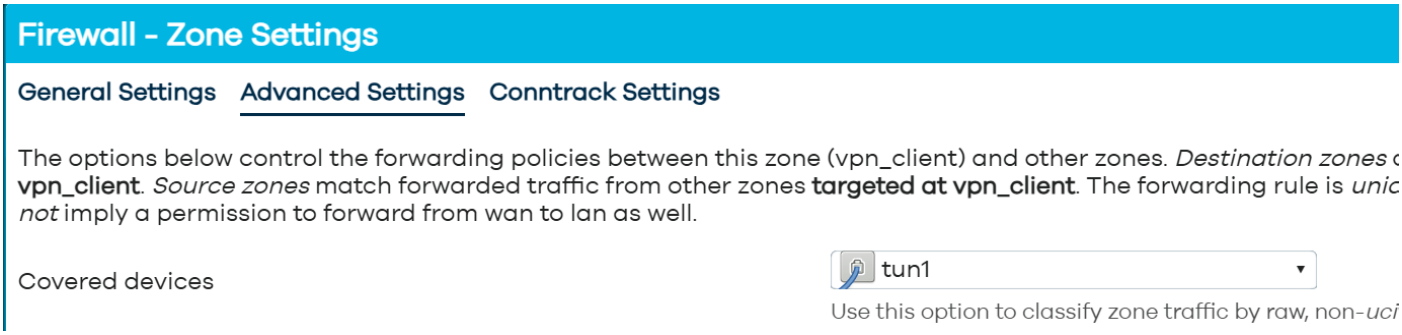
Setup Firewall

OpenVPN Client

Create a new firewall zone:
Network > Firewall > Add
Name: **vpn_client**
Input: **reject**
Output: **accept**
Intra zone forward: **reject**
IPv4 Masquerading: **Enabled** (ticked)
MSS clamping: **Enabled** (ticked)
Covered Networks: none
Allow forward from source zones: **lan**



In this example I have adapted my config file to use: dev tun1
As we have used script-security 2 and use our own config there is no device information available for the firewall to use so we have to add the device on the Advanced Settings tab under Covered devices:



For IPv6 also enable Masquerading IPv6 on the Advanced Settings Tab.

/etc/config/firewall:

```
config zone
        option name 'vpn_client'
        option input 'REJECT'
        option output 'ACCEPT'
        option forward 'REJECT'
        option masq '1'
        option mtu_fix '1'
        option masq6 '1'
        list device 'tun1'

config forwarding
        option src 'lan'
        option dest 'vpn_client'
```

Alternatively you can add an extra interface with name: *tun1_client*, protocol: *unmanaged* and device: *tun1*
You can the use this interface *tun1_client* to add under Covered networks instead of adding the device under Covered devices

OpenVPN Server

Open up firewall for the server port

Create new zone vpn_server, Allow Input, Output and Intra zone forward

Allow from VPN to LAN , from VPN to WAN if you want internet for the connected clients and Allow from LAN to VPN if you want bidirectional traffic.

Status

View traffic with: *cat /var/run/openvpn.vpn_client.status* # replace *vpn_client* with the name of your interface
Resulting config: */var/run/openvpn.<interface-name>.conf*

Management interface

```
#Add to openvpn config for management
management 127.0.0.1 14
management-log-cache 100
```

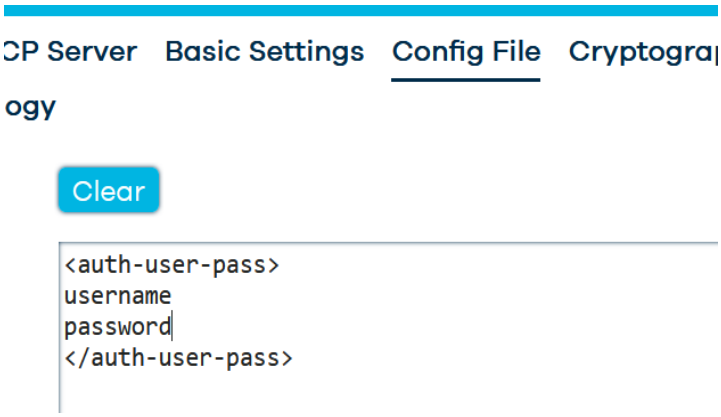
#for viewing: `nc 127.0.0.1 14`

Problems

No host route to server
When letting OpenWRT setup the routing via the hotplug scripts (script-security 3) there is no host route made, problem in /usr/libexec/openvpn-hotplug
This is solved in commit:
https://github.com/openwrt/packages/commit/c82ed824436d6e6056994267356928558 3a3e28b
If you make an IPv6 connection to the host (server) then it only works if source routing is disabled on wan6.
You can disable host routing if you add in the network config file the option:
`option nohostroute '1'`

The default route is replaced instead of using 128.0.0.0/1 and 0.0.0.0/1 which causes no default route being available when the tunnel is down and a whole *service network restart* is then necessary

auth-user-pass is not working via the GUI, you can add it in the config e.g.:
option auth_user_pass ' /etc/openvpn/<aut-user-pass-file> '
but once the GUI is saved it is gone again a good alternative is to just add it to the openvpn conifg in the GUI:



Most other missing options e.g.
pull_filter 'ignore "redirect-gateway ipv6'
remote-random
etc.
Can also be added to the config file see above

The can also be set in the network config file like
option pull_filter 'ignore "redirect-gateway ipv6'
option remote_random '1'

But I have not found those in the GUI yet

OpenVPN can use a gateway although not per se necessary but the hotplug routing does not take that into account

Addendum 1 Script to convert config to uci

```
#!/bin/sh
# ovpn2uci.sh — Convert an OpenVPN config file to OpenWrt UCI commands
#
# Usage:  ./ovpn2uci.sh <config_file> [interface_name] > /tmp/openvpn-uci-setup.sh
#         the `interface_name` bust be less than 15 characters and cannot contain a hyphen!
#         inspect the output script and execute if OK
#         restart network: `service network restart`
# Output: UCI shell commands printed to stdout; redirect to a file or pipe to sh
# Version: 20-may-2026
# Authored by: egc112 and Claude
#
# Notes:
#   • ash/busybox compatible — no bash-specific features
#   • Inline blocks (<ca>, <cert>, <key>, <tls-crypt>, …) are written to
#     temp files during processing, then emitted as heredocs pointing to
#     INLINE_DIR at runtime.
#   • Options that appear more than once are emitted with `uci add_list`.
#   • The OpenVPN `proto` option is stored as `vpn_proto` to avoid clashing
#     with the netifd `proto=openvpn` key.
#   • Unknown / unsupported options are emitted as comments so nothing is
#     silently dropped.

set -e

# ─── Configuration ──────────────────────────────────────────────────────────
CONFIG_FILE="${1:-}"
NAME="${2:-ovpn}"
OPTIONS_FILE="/usr/share/openvpn/openvpn.options"
INLINE_DIR="/etc/openvpn"
TMPDIR="/tmp/ovpn2uci_$$"

# ─── Cleanup on exit ────────────────────────────────────────────────────────
trap 'rm -rf "$TMPDIR"' EXIT
mkdir -p "$TMPDIR"

# ─── Argument validation ────────────────────────────────────────────────────
if [ -z "$CONFIG_FILE" ]; then
    echo "Usage: $0 <config_file> [interface_name]" >&2
    exit 1
fi
if [ ! -f "$CONFIG_FILE" ]; then
    echo "Error: config file '$CONFIG_FILE' not found" >&2
    exit 1
fi

# ─── Load the option type lists ─────────────────────────────────────────────
OPENVPN_PARAMS=""
OPENVPN_PARAMS_STRING=""
OPENVPN_UINTS=""
OPENVPN_BOOLS=""
OPENVPN_LIST=""
```

```
if [ -f "$OPTIONS_FILE" ]; then
    # shellcheck disable=SC1090
    . "$OPTIONS_FILE"
    # Merge all parameter-taking variants into one list
    OPENVPN_PARAMS="$OPENVPN_LIST
$OPENVPN_PARAMS_STRING
$OPENVPN_UINTS"
else
    echo "# WARNING: '$OPTIONS_FILE' not found — option validation disabled" >&2
fi

# ——— Helper functions ————————————————————————————————————————————

# Normalise an option name: hyphens → underscores, lower-case
normalize() { printf '%s' "$1" | tr '-' '_' | tr 'A-Z' 'a-z'; }

# Return true (0) when $1 exactly matches a word in newline-separated list $2
in_set() { printf '%s' "$2" | grep -qw "$1"; }

# Extract opening inline tag name from a line like <tls-crypt>; empty if no match
open_tag() { printf '%s' "$1" | sed -n 's/^<\([a-zA-Z][a-zA-Z0-9_-]*\)>$/\1/p'; }

# Extract closing inline tag name from a line like </tls-crypt>; empty if no match
close_tag() { printf '%s' "$1" | sed -n 's/^<\/\([a-zA-Z][a-zA-Z0-9_-]*\)>$/\1/p'; }

# Return option count stored in a temp file (default 0)
opt_count() {
    _safe=$(normalize "$1" | tr -c 'a-zA-Z0-9_' '_')
    _cf="$TMPDIR/count_$_safe"
    [ -f "$_cf" ] && cat "$_cf" || printf '0'
}

# ——— PASS 1 — collect inline blocks into temp files ——————————————————————
INLINE_TAGS=""   # space-separated list of tag names seen
in_block=0
cur_tag=""

while IFS= read -r line || [ -n "$line" ]; do
    otag=$(open_tag "$line")
    ctag=$(close_tag "$line")

    if [ -n "$otag" ]; then
        in_block=1
        cur_tag="$otag"
        : > "$TMPDIR/inline_$cur_tag"    # create / truncate
        continue
    fi

    if [ -n "$ctag" ] && [ "$in_block" = "1" ]; then
        in_block=0
        INLINE_TAGS="$INLINE_TAGS $cur_tag"
        cur_tag=""
        continue
    fi

    [ "$in_block" = "1" ] && printf '%s\n' "$line" >> "$TMPDIR/inline_$cur_tag"
done < "$CONFIG_FILE"
```

```
# ——— PASS 2 — count occurrences of each option (to detect repeats) ——————————
in_block=0

while IFS= read -r line || [ -n "$line" ]; do
    # skip blank lines and comments
    case "$line" in "|'#'*) continue ;; esac
    # also skip lines that are only whitespace
    stripped=$(printf '%s' "$line" | tr -d '[:space:]')
    [ -z "$stripped" ] && continue

    otag=$(open_tag "$line")
    if [ -n "$otag" ]; then in_block=1; continue; fi
    ctag=$(close_tag "$line")
    if [ -n "$ctag" ]; then in_block=0; continue; fi
    if [ "$in_block" = "1" ]; then continue; fi

    raw=$(printf '%s' "$line" | awk '{print $1}')
    norm=$(normalize "$raw")
    # Sanitize: replace chars not valid in a filename (e.g. / in base64) with _
    safe=$(printf '%s' "$norm" | tr -c 'a-zA-Z0-9_' '_')
    cf="$TMPDIR/count_$safe"
    if [ -f "$cf" ]; then
        c=$(cat "$cf")
        printf '%d' "$((c + 1))" > "$cf"
    else
        printf '1' > "$cf"
    fi
done < "$CONFIG_FILE"

# ——— Build skip list from inline tags ———————————————————————————————————
SKIP_OPT=""
for tag in $INLINE_TAGS; do
    SKIP_OPT="$SKIP_OPT $(normalize "$tag")"
done

# ——— Emit UCI header
————————————————————————————————————————————————————————————————————————
printf '#!/bin/sh\n'
printf '# Generated by ovpn2uci.sh from: %s\n\n' "$CONFIG_FILE"

printf "name='%s'\n\n" "$NAME"
printf "uci set network.\${name}='interface'\n"
printf "uci set network.\${name}.proto='openvpn'\n"

# ——— Emit inline blocks
———————————————————————————————————————————————————————————————————————
if [ -n "$INLINE_TAGS" ]; then
    printf '\n# -- Inline certificate / key blocks ----------------------------------\n'
    for tag in $INLINE_TAGS; do
        norm=$(normalize "$tag")
        case "$tag" in
            ca|cert|extra_certs) ext="crt" ;;
            key)                 ext="key" ;;
            *)                   ext="pem" ;;
        esac
        fpath="${INLINE_DIR}/${NAME}_${norm}.${ext}"
        printf "\ncat > '%s' << 'OVPN_INLINE_EOF'\n" "$fpath"
```

```
        cat "$TMPDIR/inline_$tag"
        printf 'OVPN_INLINE_EOF\n'
        printf "uci set network.\${name}.%s='%s'\n" "$norm" "$fpath"
    done
fi

# ─── PASS 3 — emit UCI for every regular option ──────────────────────────────
printf '\n# -- OpenVPN options ------------------------------------------------\n'
in_block=0

while IFS= read -r line || [ -n "$line" ]; do

    # Skip blank lines and full-line comments
    case "$line" in "|'#'*) continue ;; esac
    stripped=$(printf '%s' "$line" | tr -d '[:space:]')
    [ -z "$stripped" ] && continue

    # Track inline blocks
    otag=$(open_tag "$line")
    if [ -n "$otag" ]; then in_block=1; continue; fi
    ctag=$(close_tag "$line")
    if [ -n "$ctag" ]; then in_block=0; continue; fi
    if [ "$in_block" = "1" ]; then continue; fi

    # Split keyword and value
    raw=$(printf '%s' "$line" | awk '{print $1}')
    norm=$(normalize "$raw")
    # Value = everything after keyword, left-trimmed
    value=$(printf '%s' "$line" | sed "s/^${raw}//" | sed 's/^[[:space:]]*//')

    # Skip options already covered by an inline block
    case " $SKIP_OPT " in
        *" $norm "*)
            printf '# Skipped (%s set by inline block above)\n' "$raw"
            continue
            ;;
    esac

    # Special case: `proto` clashes with netifd proto=openvpn
    if [ "$norm" = "proto" ]; then
        printf '# NOTE: openvpn proto stored as vpn_proto to avoid netifd conflict\n'
        printf "uci set network.\${name}.vpn_proto='%s'\n" "$value"
        continue
    fi

    # Special case: `remote [host] [port]`
    # With one parameter it is the remote host; with two the second is the port.
          # might need to remove this
    if [ "$norm" = "remote" ]; then
        r_host=$(printf '%s' "$value" | awk '{print $1}')
        r_port=$(printf '%s' "$value" | awk '{print $2}')
        c=$(opt_count "$norm")
        if [ "$c" -gt 1 ]; then
            printf "uci add_list network.\${name}.remote='%s'\n" "$r_host"
        else
            printf "uci set network.\${name}.remote='%s'\n" "$r_host"
        fi
        if [ -n "$r_port" ]; then
```

```
            printf "uci set network.\${name}.port='%s'\n" "$r_port"
        fi
        continue
    fi

    # Classify and emit
    if in_set "$norm" "$OPENVPN_BOOLS"; then
        # Boolean flag — presence means true
        printf "uci set network.\${name}.%s='1'\n" "$norm"

    elif in_set "$norm" "$OPENVPN_LIST"; then
        # List type (e.g. data_ciphers) — one add_list per occurrence
        printf "uci add_list network.\${name}.%s='%s'\n" "$norm" "$value"

    elif in_set "$norm" "$OPENVPN_PARAMS"; then
        # Regular param — use add_list if option appears more than once
        c=$(opt_count "$norm")
        if [ "$c" -gt 1 ]; then
            printf "uci add_list network.\${name}.%s='%s'\n" "$norm" "$value"
        else
            printf "uci set network.\${name}.%s='%s'\n" "$norm" "$value"
        fi

    else
        # Unknown — preserve as comment so nothing is silently lost
        printf '# Unknown/unsupported option (skipped): %s\n' "$line"
    fi

done < "$CONFIG_FILE"

printf '\nuci commit network\n'
```

$trusted_ip : host=server-ip

```
proto_add_ipv4_route "$trusted_ip" 32 $route_net_gateway
```