Make Your Own VPN—FreeBSD, Wireguard, IPv6 and Ad-blocking Included

BY STEFANO MARINELLI

Note: This article assumes a setup based on FreeBSD. If you prefer a version based on OpenBSD, it is available here.

VPNs are a fundamental tool for securely connecting to your own servers and devices. Many people use commercial VPNs for various reasons, ranging from not trusting their provider (especially when connecting from a public hotspot) to wanting to "go out" on the Internet with a different IP address, perhaps from another country. Here, I want to highlight some of the new features that have been brought into the base stack—many of which are enabled by default, and some of which may need to be specifically activated. Each feature will be described with details that may help improve the networking experience.

Whatever the reason, solutions are not lacking. I have always set up management VPNs to allow servers and/or clients to communicate with each other using secure channels. Lately, I have been activating IPv6 connectivity on all my devices (both desktop/servers and mobile devices) and I needed to quickly create a node that concentrated some networks and allowed them to go out on the network in IPv6. The tools I used and will describe are:

I want to highlight some of the new features that have been brought into the base stack.

- VPS – in this case, I used a basic Hetzner Cloud VPS, but any provider that provides IPv6 connectivity will do – if you want IPv6, of course.
- FreeBSD – a versatile, stable, and secure operating system.
- Wireguard – lightweight, secure, and at the same time, not very "chatty," so it is also gentle on mobile device batteries. When there is no traffic, it simply does not transmit/receive anything. Well supported by all major desktop and server operating systems as well as Android and iOS devices.
- Unbound – can make DNS queries directly to root servers, not through forwarders. It also allows you to insert block-lists and have a result similar to that of Pi-Hole (i.e., ad-blocking).
- SpamHaus lists – to immediately stop connections to and from users on blacklists.

The first step is to activate a VPS and install FreeBSD. On the Hetzner Cloud console, there might not be a pre-built FreeBSD image, but only a selection of Linux distributions. Don't worry, just choose any of them and create the VPS. Once done, the FreeBSD ISO image will be available among the "ISO Images." Just insert the virtual CD, restart the VPS, and the FreeBSD installation will appear in the console.

I won't go into detail, the operation is simple and straightforward. The only precaution (in the case of a Hetzner Cloud VPS) is to use "DHCP" for IPv4 but, for now, do not configure IPv6. It will be configured later.

Install all FreeBSD updates (using the **`freebsd-update fetch install`** command) and reboot.

Wireguard, on FreeBSD, is now available as a kernel module and the userland can be installed using the **`pkg install wireguard-tools`** package manager. This means you can easily keep it updated alongside other software on the system.

The first step is to configure IPv6 on the VPS. In the case of Hetzner, unfortunately, they only provide a /64, so it will be necessary to segment the assigned network. In this example, it will be divided into /72 subnetworks - to find valid subclasses, it will be possible to use a calculator.

The `/etc/rc.conf` file should have entries similar to:

```
ifconfig_vtnet0="DHCP"
ifconfig_vtnet0_ipv6="inet6 2a01:4f8:cafe:cafe::1 prefixlen 72"
ipv6_defaultrouter="fe80::1%vtnet0"
```

In short, keep the base address assigned by Hetzner, but change the prefix length to 72 - thus giving the possibility of having other networks available.

It is now necessary to enable forwarding for IPv4 and IPv6. Add these lines to the `/etc/sysctl.conf` file:

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

After reboot, test it:

```
ping6 google.com
```

If everything has been configured correctly, the ping will be executed and google.com will reply.

To configure Wireguard, a few steps will be necessary. First of all, the private key will need to be created:

```
wg genkey | tee /dev/stderr | wg pubkey  | grep --label PUBLIC -H .
```

You will get a private key and a public key. Take note of the public key — it will be needed to configure the clients.

Now create a new file called `/usr/local/etc/wireguard/wg0.conf`:

```
[Interface]
Address = 172.14.0.1/24,2a01:4f8:cafe:cafe:100::1/72
ListenPort = 51820
PrivateKey = YUkS6cNTyPbXmtVf/23ppVW3gX2hZIBzlHtXNFRp80w=
```

A new Wireguard interface called `wg0` is being created. Start the Wireguard interface:

```
service wireguard enable
sysrc wireguard_interfaces="wg0"
service wireguard start
```

If everything has been entered correctly, the interface should come up. Check its status:

```
wg
```

As for the firewall, FreeBSD comes with no `pf` configuration. In my setups, I tend to block what is not needed and be permissive with what may be useful. However, I like to keep out the "bad guys," so I use blacklists. **pf** allows elements to be inserted and removed from tables at runtime, so the firewall can be configured accordingly.

To download and apply the Spamhaus lists, I use a simple but effective script found on the Internet, but for OpenBSD.

For the Spamhaus lists, continue with the FreeBSD script creation.

Create the script in **/usr/local/sbin/spamhaus.sh**:

```
#!/bin/sh
#
#this is normally run once per day via cron.
#
echo updating Spamhaus DROP lists:
(
  { fetch -o - https://www.spamhaus.org/drop/drop.txt && \
    fetch -o - https://www.spamhaus.org/drop/edrop.txt && \
    fetch -o - https://www.spamhaus.org/drop/dropv6.txt ; \
  } 2>/dev/null | sed "s/;/#/" > /var/db/drop.txt
)
pfctl -t spamhaus -T replace -f /var/db/drop.txt
```

Make it executable and run it. Pf isn't enabled, so you'll get an error — but this will create the **/var/db/drop.txt** file:

```
chmod a+rx /usr/local/sbin/spamhaus.sh
/usr/local/sbin/spamhaus.sh
```

There are many possibilities to configure **pf** on FreeBSD. A fairly simple example could be this:

```
ext_if="vtnet0"
wg0_if="wg0"
wg0_networks="172.14.0.0/24"

set skip on lo

nat on $ext_if from { $wg0_networks } to any -> ($ext_if)

# Spamhaus DROP list:
table <spamhaus> persist file "/var/db/drop.txt"

block drop log quick from <spamhaus>

# Pass ICMP on ipv6
```

```
pass quick proto ipv6-icmp
# Block from ipv6 to wg0 network
block in quick on $ext_if inet6 to { 2a01:4f8:cafe:cafe:100::/72 }
# Pass Wireguard traffic - in and out
pass quick on $wg0_if

# default deny
block in
block out

pass in on $ext_if proto tcp to port ssh
pass in on $ext_if proto udp to port 51820

pass out on $ext_if
```

This is a very simple configuration: it blocks everything that is present in the list downloaded from Spamhaus, allows NAT from the Wireguard network to the public interface, allows ICMP traffic in IPv6 (necessary for the network to function properly) while blocking incoming traffic to the Wireguard IPv6 LAN (remember that the IPs will be public and directly reachable, so we don't want to expose our devices by default). All traffic on the Wireguard interface will be allowed to pass. Then everything will be blocked and exceptions will be specified, i.e., allowing SSH and Wireguard connections (of course). Authorization will also be granted to allow traffic to exit from the public network interface.

Save this configuration to `/etc/pf.conf`.

Enable and start `pf`:

```
service pf enable
service pf start
```

You will probably be kicked out of the system. Don't worry, just reconnect. pf is doing its job.

If everything went correctly, the firewall should have loaded the new rules.

To obtain caching of DNS queries and the related ad-block, it is now time to configure Unbound. Let's install it with:

```
pkg install unbound
```

A while ago, I found a script which I slightly adapted. I don't remember where I got it, so I'll paste it here without citing the original creator.

Create a script to update the unbound ad-block, in `/usr/local/sbin/unbound-ad-hosts.sh`:

```
#!/bin/sh
#
# Using blacklist from pi-hole project https://github.com/pi-hole/
# to enable AD blocking in unbound(8)
#
PATH="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin"
```

```
# Available blocklists - comment line to disable blocklist
_disconad="https://s3.amazonaws.com/lists.disconnect.me/simple_ad.txt"
_discontrack="https://s3.amazonaws.com/lists.disconnect.me/simple_tracking.
txt"
_stevenblack="https://raw.githubusercontent.com/StevenBlack/hosts/master/
hosts"

# Global variables
_tmpfile="$(mktemp)" && echo '' > $_tmpfile
_unboundconf="/usr/local/etc/unbound/unbound-adhosts.conf"

# Remove comments from blocklist
simpleParse() {
  fetch -o - $1 | \
  sed -e ‘s/#.*$//’ -e ‘/^[[:space:]]*$/d’ >> $2
}

# Parse DisconTrack
[ -n "${_discontrack}" ] && simpleParse $_discontrack $_tmpfile

# Parse DisconAD
[ -n "${_disconad}" ] && simpleParse $_disconad $_tmpfile

# Parse StevenBlack
[ -n "${_stevenblack}" ] && \
  fetch -o - $_stevenblack | \
  sed -n '/Start/,$p' | \
  sed -e 's/#.*$//' -e '/^[[:space:]]*$/d' | \
  awk '/^0.0.0.0/ { print $2 }' >> $_tmpfile

# Create unbound(8) local zone file
sort -fu $_tmpfile | grep -v "^[[:space:]]*$" | \
awk '{
  print "local-zone: \"" $1 "\" redirect"
  print "local-data: \"" $1 " A 0.0.0.0\""
}' > $_unboundconf && rm -f $_tmpfile

service unbound reload 1>/dev/null

exit 0
```

After saving the script, make it executable and run it:

```
chmod a+rx /usr/local/sbin/unbound-adhosts.sh
/usr/local/sbin/unbound-adhosts.sh
```

Now, the Unbound configuration file in `/usr/local/etc/unbound/unbound.conf` can be modified as follows:

```
server:
        verbosity: 1
        log-queries: no
        num-threads: 4
        num-queries-per-thread: 1024
        interface: 127.0.0.1
        interface: 172.14.0.1
        interface: 2a01:4f8:cafe:cafe:100::1
        interface: ::1
        outgoing-range: 64
        chroot: ""

        access-control: 0.0.0.0/0 refuse
        access-control: 127.0.0.0/8 allow
        access-control: ::0/0 refuse
        access-control: ::1 allow
        access-control: 172.14.0.0/24 allow
        access-control: 2a01:4f8:cafe:cafe:100::/72 allow

        hide-identity: yes
        hide-version: yes
        auto-trust-anchor-file: "/usr/local/etc/unbound/root.key"
        val-log-level: 2
        aggressive-nsec: yes
        prefetch: yes
        username: "unbound"
        directory: "/usr/local/etc/unbound"
        logfile: "/var/log/unbound.log"
        use-syslog: no
        pidfile: "/var/run/unbound.pid"
        include: /usr/local/etc/unbound/unbound-adhosts.conf

remote-control:
        control-enable: yes
        control-interface: /var/run/unbound.sock
```

Now, enable and start unbound:

```
service unbound enable
service unbound start
```

If everything has been set up correctly, unbound will be able to respond to DNS requests made on `172.14.0.1` and `2a01:4f8:cafe:cafe:100::1`.

Now it is possible to configure the Wireguard client. Create a new configuration by inserting "172.14.0.2/32, 2a01:4f8:cafe:cafe:100::2/128" (the ones that will later be entered in the peer configuration of the server) in the local IP addresses. Set the DNS server address to

"172.14.0.1" and/or its corresponding IPv6 address (in the example, 2a01:4f8:cafe:cafe:100::1 - yours will be different). In the peer section, insert the server's data, including its public key, IP address:port (in the example, the port is 51820), and allowed addresses (setting "0.0.0.0/0, ::0/0" means "all connections will be sent via Wireguard" — all the traffic will pass through the VPN for both IPv4 and IPv6).Each implementation has its own procedure (Android, iOS, MikroTik, Linux, etc.) but essentially it is sufficient to create the right configuration both on the server and on the client.

Reopen the Wireguard configuration file **`/usr/local/etc/wireguard/wg0.conf`** and add:

```
[Interface]
Address = 172.14.0.1/24,2a01:4f8:cafe:cafe:100::1/72
ListenPort = 51820
PrivateKey = YUkS6cNTyPbXmtVf/23ppVW3gX2hZIBzlHtXNFRp80w=

[Peer]
PublicKey = *client's public key*
AllowedIPs = 172.14.0.2/32, 2a01:4f8:cafe:cafe:100::2/128
```

The client's public key will be shown by the client itself.

Reload the Wireguard configuration:

```
service wireguard restart
```

It is also possible to use the VPN only as an ad-blocker, by only routing DNS traffic through it. To achieve this result, configure the client so that the only allowed address is the one of the just-configured unbound (in this example, 172.14.0.1 and/or 2a01:4f8:cafe:cafe:100::1) — DNS resolution will occur via VPN, but browsing will continue to work through the main provider.

To automatically update the spamhaus and ad-block lists, we will use cron.First, create a script, for example, **`/usr/local/sbin/update-blocklists.sh`**:

```
#!/bin/sh

/usr/local/sbin/unbound-adhosts.sh
/usr/local/sbin/spamhaus.sh
```

Make it executable:

```
chmod +x /usr/local/sbin/update-blocklists.sh
```

Then, add it to the crontab to run daily:

```
echo "@daily /usr/local/sbin/update-blocklists.sh" >> /etc/crontab
```

This approach benefits from both update management and security perspectives.

STEFANO MARINELLI is an IT Consultant with over two decades of experience in the realms of IT consulting, training, research, and publishing. His expertise spans across operating systems, with a special emphasis on *BSD systems — FreeBSD, NetBSD, OpenBSD, DragonFlyBSD - and Linux. Stefano is also the barista at BSD Cafe, a vibrant community hub for *BSD enthusiasts, and has led the FreeOsZoo project at the University of Bologna, making open-source operating system images accessible for virtual machines.