Tor-WLAN with Raspberry Pi

Thoughts upfront

The following is not for a high-level of anonymity and privacy; if anonymity is really crucial please look for more secure solutions like Tails or Whonix instead. Using Tor will effectively only hide your IP address, i.e. no-one can see your traffic as it leaves your location, however, starting from the Tor Exit the IP traffic looks the same as without Tor – just coming from the Tor-Exit and not from your IP address directly. Any private data included in your traffic itself will still be visible (to everyone if not encrypted, i.e. using http and the website provider if using https encrypted traffic). Keep in mind that although many sites (seem) to use https they are in fact hosted by companies like Cloudflare which terminated the https tunnel and have full access to everything sent to the website. Also your specific browser can be identified easily (although not related to you in person) via so-called fingerprinting principles.

What you will need: Raspberry Pi 3 (comes with WLAN included), a LAN cable, a SD card and a computer to prepare the SD card and then configure the Raspberry Pi. How to setup the Raspberry Pi as a mini-server (i.e. headless, without a display) is described in a separate article and is a prerequisite for the following.

Enable WLAN (Access Point with DHCP)

First, we shutdown the WLAN interface while we configure it in the next steps:

```
sudo ifdown wlan0
```

Disable the DHCP Client on WLAN

Normally the dhcpcd daemon (DHCP client) will search the network for a DHCP server to assign a IP address to wlan0. This is disabled by editing the configuration file:

```
sudo nano /etc/dhcpcd.conf
```

We tell the DHCP client that on the wlan0 interface we have a static IP address and it should not configure anything else on it. This is achieved by adding at the very end of the config file:

```
interface wlan0
    static ip_address=192.168.200.1/24
    nohook wpa_supplicant
```

and then restart the service so the new config is loaded:

```
sudo systemctl restart dhcpcd
```

Enable the Access-Point Server

Install the WiFi Access-Point server:

```
sudo apt install hostapd
```

Create a new config file for the access point:

```
sudo nano /etc/hostapd/hostapd.conf
```

Paste the below – and change the country code, the WLAN name (SSID) and the WLAN password before saving the file:

```
interface=wlan0
driver=nl80211
country_code=FR
ssid=Your-WLAN-Name
hw_mode=g
channel=6
```

```
ieee80211n=1
wmm_enabled=1
ht_capab=[HT40][SHORT-GI-20][DSSS_CCK-40]
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=Your-WLAN-Password
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

The channel could be set to either 1, 6 or 11 depending on other wireless networks in your area. If you don’t know just take one by chance. Next, point the server to the right config file:

```
sudo nano /etc/default/hostapd
```

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

And then ensure the service is started:

```
sudo systemctl start hostapd
sudo systemctl status hostapd
sudo systemctl enable hostapd
```

Enable the DHCP Server

Start out by installing dnsmasq with the following command

```
sudo apt install -y dnsmasq
```

and then make a backup of the initial standard configuration for later reference before enabling the new config:

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.backup
```

We will put the DHCP configuration in the file dnsmasq.dhcp.conf and edit it with:

```
sudo nano /etc/dnsmasq.d/dnsmasq.dhcp.conf
```

The new configuration should be something similar to the following, however with the ip address information and things like the MAC addresses changed to your local setup:

```
# Configuration file for dnsmasq (DHCP part)

# Extra logging for DHCP: log all the options sent to DHCP clients and the tags
# used to determine them.
#log-dhcp

# Suppress logging of the routine operation of these protocols. Errors and problems
# will still be logged. quiet-dhcp and quiet-dhcp6 are over-ridden by log-dhcp.
quiet-dhcp

# This is our local network to be served as DHCP on wlan0
dhcp-range=wlan0,192.168.200.20,192.168.200.199,255.255.255.0,24h

# This is the only DHCP-server for wlan0
dhcp-authoritative

# This is your local domain name. It will tell the DHCP server which
# host to give out IP addresses for.
domain=alpha

# Set the Gateway IP address
dhcp-option=option:router,192.168.200.1

# Set the DNS server
dhcp-option=option:dns-server,192.168.200.1

# Set the NTP time server address
dhcp-option=option:ntp-server,192.168.200.1

# adapted for a typical dnsmasq installation where the host running
# dnsmasq is also the host running samba.
```

```
# you may want to uncomment some or all of them if you use
# Windows clients and Samba.
dhcp-option=19,0            # option ip-forwarding off for
clients
dhcp-option=44,0.0.0.0       # set netbios-over-TCP/IP
nameserver(s) aka WINS server(s)
dhcp-option=45,0.0.0.0      # netbios datagram distribution
server
dhcp-option=46,8           # netbios node type


# Always allocate the same DHCP IP address based on the MAC
address
dhcp-host=00:25:a5:34:af:e7, Ethernet-Bridge,  192.168.200.20,
infinite
dhcp-host=84:cf:bf:89:9b:fd, Mobile,        192.168.200.120,
infinite
```

To test if the config for dnsmasq is free of errors check the syntax with:

```
sudo dnsmasq --test
```

DNS Server and Cache

We put the DNS server configuration in a separate file (both DHCP and DNS is the same dnsmasq software and the config could well be in one file; but it’s a bit easier to read that way):

```
sudo nano /etc/dnsmasq.d/dnsmasq.dns.conf
```

and paste in the following

```
# Configuration file for dnsmasq (DNS part)

# For debugging, log each DNS query as it passes through
dnsmasq.
#log-queries

# Listen on these IP addresses for DNS requests (always
include localhost):
```

```
listen-address=127.0.0.1,192.168.200.1,10.80.0.1

# Where to send DNS request to which can't be resolved locally.
# Here we use the local Tor service listening on port 5300 (see /etc/tor/torrc)
server=127.0.0.1#5300

# The bind-interfaces directive instructs dnsmasq to bind only to
# the network interface specified in the listen-address directive.
bind-interfaces

# The no-hosts directive instructs dnsmasq not to read hostnames
# from /etc/hosts.
no-hosts

# Read hostname information from this file in addition
addn-hosts=/etc/dnsmasq.hosts

# Never forward plain names (without a dot or domain part)
upstream
domain-needed

# Never forward addresses in the non-routed address spaces.
bogus-priv

# If you don't want dnsmasq to read /etc/resolv.conf or any other
# file, getting its servers from this file instead, then uncomment this.
no-resolv

# If you don't want dnsmasq to poll /etc/resolv.conf or other resolv
# files for changes and re-read them then uncomment this.
no-poll

# Set this if you want to have a domain
```

```
# automatically added to simple names in a hosts-file.
expand-hosts

# Number of hostnames being cached in RAM for DNS lookups
cache-size=10000

# Do not cache negative responses
no-negcache

# Resolve these domains locally only, don't send upstream
local=/alpha/
local=/local/

# Some simple domain blocking (doesn't block the IP, just domain-name)
address=/doubleclick.net/0.0.0.0
address=/google-analytics.com/0.0.0.0
address=/analytics.google.com/0.0.0.0
address=/amazon-adsystem.com/0.0.0.0
address=/analytics.yahoo.com/0.0.0.0
address="cn"0.0.0.0
```

To resolve the hostname of this server we need to provide it in a separate host-file:

```
sudo nano /etc/dnsmasq.hosts
```

and just list the local hostname and its IP address on the WLAN:

```
192.168.200.1   MyTorGateway
```

so can use its name on other devices on the WLAN (instead of the IP address).

To test if the above config for dnsmasq is free of errors run:

```
sudo dnsmasq --test
```

If everything is fine, start the dnsmasq service and check its status with:

```
sudo service dnsmasq start
```

sudo service dnsmasq status

To see the listening ports, check with:

sudo netstat -tulpn | grep dnsmasq

Enable a Time Server (NTP via chrony)

Since the aim is to route all traffic from the wireless network through Tor we need to provide the time to the WLAN since this can’t be done over Tor. The best ntp server for our purpose seems to be chrony. It’s installed via:

sudo apt install -y chrony

and then we configure it (after saving the initial config for later reference):

```
sudo mv /etc/chrony/chrony.conf /etc/chrony/chrony.conf.default
sudo nano /etc/chrony/chrony.conf
```

and paste in the following:

```
# Welcome to the chrony configuration file. See chrony.conf(5) for more
# information about usuable directives.

# The time server where we get our time from; here the internet router
server 192.168.178.1 iburst
#pool pool.ntp.org iburst

# Time server for the following subnet
allow 192.168.200.0/24

# Listen on this interface for client ntp requests
bindaddress 192.168.200.1
```

```
# This directive specify the file into which chronyd will store the rate
# information.
driftfile /var/lib/chrony/chrony.drift

# Uncomment the following line to turn logging on.
#log tracking measurements statistics

# Log files location.
logdir /var/log/chrony

# The hardware clock (rtc) is always UTC
rtconutc

# This directive enables kernel synchronisation of the real-time clock.
rtcsync

# Step the system clock instead of slewing it if the adjustment is larger than
# one second, but only in the first three clock updates.
makestep 1 3
```

If the devices on the WLAN don’t pick up time after a few minutes it might be necessary to list the ntp server 192.168.200.1 in the /etc/ntp.conf (or similar) config files of those devices.

Data Forwarding (WLAN ↔ LAN)

Only in case you want to access servers on your LAN (the one directly attached to the internet) from the Raspberry WLAN you need to allow IP4 forwarding on the Raspberry:

```
sudo nano /etc/sysctl.conf
```

Set the forwarding option to 1:

```
net.ipv4.ip_forward=1
```

and then reload the configuration:

```
sudo sysctl -p
```

You can double-check that it worked with:

```
sudo sysctl net.ipv4.ip_forward
```

The WLAN is now setup and we only need to enable the interface:

```
sudo ifconfig wlan0 up
sudo reboot
```

Now you should test it – you should be able to connect to the new WLAN just defined and if IP4 forwarding was enabled you should have access to the Internet (no Tor yet though).

Install and configure Tor

Getting Tor up and running is really easy (including the recommended package to keep the key updated):

```
sudo apt install -y tor
```

Keep the initial config file for reference before changing it:

```
sudo mv /etc/tor/torrc /etc/tor/torrc.default
sudo nano /etc/tor/torrc
```

```
Log notice file /var/log/tor/tor.log
AvoidDiskWrites 1

ExitRelay 0
ExitPolicy reject *:*

TransPort      127.0.0.1:9040      IsolateClientAddr
IsolateClientProtocol IsolateDestAddr IsolateDestPort
DNSPort   127.0.0.1:5300
TransPort    192.168.200.1:9040    IsolateClientAddr
IsolateClientProtocol IsolateDestAddr IsolateDestPort
DNSPort   192.168.200.1:5300
TransPort      10.80.0.1:9040      IsolateClientAddr
IsolateClientProtocol IsolateDestAddr IsolateDestPort
```

```
DNSPort   10.80.0.1:5300

AutomapHostsOnResolve 1
AutomapHostsSuffixes .onion,.exit
VirtualAddrNetworkIPv4 10.12.0.0/16

ExitNodes     9EAD5B2D3DBD96DBC80DCE423B0C345E920A758D,
85D4088148B1A6954C9BFFFCA010E85E0AA88FF0
```

Check that Tor is happy with the config file and there are no issues with it:

```
sudo -u debian-tor tor --verify-config
```

The last line should say that everything looks fine. Then reload the new Tor config:

```
sudo systemctl reload tor.service
```

Check the logs for what Tor does and if it complains about anything – the following commands might be useful to check for any errors:

```
sudo systemctl status tor.service
sudo cat /var/log/tor/tor.log
journalctl | grep 'Tor'
```

In case of issues with tor one could increase the level of logging temporarily – in the tor config file /etc/tor/torrc replace the ‚log …‘ configuration with debug, info, notice, warn, or err. For example a ‚log info…‘ statement will provide more details. In the long run the log-level of ’notice‘ should be the best choice.

If anonymity and privacy is not your major concern but rather useability you might want to specify one or more Exit nodes to be used for all your traffic. This will ensure for example that you can specify the country where you seem to be coming from. To do that, just add to torrc config file something like:

```
ExitNodes 85D4088148B1A6954C9BFFFCA010E85E0AA88FF0
```

Enable the Firewall

First install the firewall frontend and enable the firewall:

```
sudo apt install nftables -y
sudo systemctl enable nftables.service
```

Enable the following firewall rules, starting with a config file in your home directory

```
nano ~/nftables.conf
```

and paste in

```
#!/usr/sbin/nft -f

flush ruleset

define wireguard_port = 40042

table ip filter {
        chain INPUT {
                  type filter hook input priority 0; policy
drop;
                iif lo accept
                ct state related,established accept
                ct state invalid drop
                 ip saddr {192.168.178.0/24, 192.168.200.0/24}
tcp dport ssh ct state new accept
                 ip saddr {192.168.178.0/24, 192.168.200.0/24}
icmp type echo-request accept
                      udp dport $wireguard_port accept
# might not be needed
                 iifname {"wlan0", "wg0"} udp dport domain
accept
                 iifname {"wlan0", "wg0"} tcp dport domain
accept
                  iifname {"wlan0", "wg0"} udp dport {ntp,
bootps} accept
                iifname {"wlan0", "wg0"} tcp dport 9040 accept
        }
```

```
        chain FORWARD {
                 type filter hook forward priority 0; policy
drop;
                ct state related,established accept
                ct state invalid drop
                  iifname "wlan0" ip daddr 192.168.178.0/24
accept
        }

        chain OUTPUT {
                 type filter hook output priority 0; policy
drop;
                oif lo accept
                ct state related,established accept
                ct state invalid drop
                skuid "debian-tor" accept
                udp dport ntp accept
                ip daddr 127.0.0.1 accept
                 ip daddr {192.168.178.0/24, 192.168.200.0/24,
255.255.255.255} accept
        }
}

table ip nat {
        chain PREROUTING {
                type nat hook prerouting priority -100; policy
accept;
                 iifname "wlan0" udp dport domain redirect to
:53
                 iifname "wlan0" tcp dport domain redirect to
:53
                 iifname "wlan0" udp dport ntp    redirect to
:123
                  iifname "wlan0" ip daddr {192.168.178.0/24,
192.168.200.0/24} accept
                 iifname "wlan0" tcp flags & (fin|syn|rst|ack)
== syn redirect to :9040
                iifname "wg0" udp dport domain redirect to :53
                iifname "wg0" tcp flags & (fin|syn|rst|ack) ==
syn redirect to :9040
        }
```

```
        chain INPUT {
                  type nat hook input priority 100; policy
accept;
        }

        chain POSTROUTING {
                type nat hook postrouting priority 100; policy
accept;
                  oifname "eth0" ip saddr 192.168.200.0/24
masquerade
        }

        chain OUTPUT {
                  type nat hook output priority -100; policy
accept;
                skuid "debian-tor" accept
                 ip daddr {192.168.178.0/24, 192.168.200.0/24}
accept
                 tcp flags & (fin|syn|rst|ack) == syn redirect
to :9040
        }
}

table ip6 filter {
        chain INPUT {
                  type filter hook input priority 0; policy
drop;
                iif lo counter accept
        }

        chain FORWARD {
                  type filter hook forward priority 0; policy
drop;
        }

        chain OUTPUT {
                  type filter hook output priority 0; policy
drop;
                oif lo counter accept
                counter reject
        }
```

}

and activate these firewall rules with

sudo nft -f nftables.conf

In case something goes horribly wrong (e.g. you lock ssh sessions) you can hard reboot the server and will start without the firewall rules.

To list the active rules applied by nft currently just check with:

sudo nft list ruleset

Note that nft uses its own matching of service names to port numbers – to see the list simply type in:

nft describe tcp dport

Once you're happy with them working make them permanent with copying them to the standard place (enabled on reboot):

mv ./nftables.conf /etc/nftables.conf

Note on Firefox configuration

If you're using firefox as browser you also need to change its config as by standard firefox is blocking access to services residing on tor (onion services). To do so, type

about:config

in the field at the top (where you would type in e.g. a web-address usually) then search for „onion" at the top and change the value for network.dns.blockDotOnion to „false" by a double-click on it. Btw, Chrome and Edge aren't blocking tor by default and they will work right away with the standard configuration. Test access to onion-services by browsing to e.g. „https://protonirockerxow.onion".

If you don’t use WebRTC services then it’s strongly recommended to disable media.peerconnection.enabled in about:config. Otherwise your local IP address will be visible to everyone on the internet (also while connected via Tor).

It might be good to also minimize the tracking via fingerprinting. One example is to enable privacy.resistfingerprinting by setting to true. Also one should disable plugin.expose_full_path and dom.battery.enabled by setting them both to false.

Optional but highly recommended is to install the „Privacy Pass“ Firefox add-on; it helps to avoid many of the annoying challenges one has to complete when using Tor. Additionally, and also optional it might be wise to install the „uBlock Origin“ add-on as well to avoid advertisements, malware, etc. which would otherwise cause additional traffic and would increases tracking capabilities.

Additional recommended add-ons for Firefox are: „Cloud Firewall“, „HTTPS Everywhere“, „Privacy Badger“, and „ClearURLs“.

Some Improvements and Hardening

IP Forwarding WLAN ↔ LAN

If no connection between WLAN and LAN is required one should remove IPv4 forwarding – this can be done in two ways (one is sufficient, two are safer):

sudo nano /etc/sysctl.conf

net.ipv4.ip_forward=0

sudo sysctl -p

or remove the rule for the forwarding filter rule of ferm.

Add the official Tor Repository

We add the official repository on the Tor network of the tor-project for Debian (as outlined at www.torproject.org/docs/debian.html.en):

sudo nano /etc/apt/sources.list.d/torproject.org.list

and put in the following:

```
# Tor onion-site replacement for:
# deb http://deb.torproject.org/torproject.org stretch main

deb http://sdscoq7snqtznauu.onion/torproject.org stretch main
```

However, Debian is not supporting anonymous and safe software repos by standard, so we need to enable it first – create a new apt config file:

sudo nano /etc/apt/apt.conf.d/80onion-repos

and put in just one line:

```
Acquire::BlockDotOnion "false";
```

Now finally update and upgrade the system with:

```
sudo apt update
sudo apt upgrade -y
```

Automatic updates for Tor

It’s also recommended to get the tor software upgraded automatically. Assuming that the package ‚unattended-upgrades‘ is installed already, we simply need to add the updates from the torproject to the config:

sudo nano /etc/apt/apt.conf.d/50unattended-upgrades

and add the line with the TorProject repository:

```
"origin=Debian,codename=${distro_codename},label=Debian-Security";
        "origin=TorProject,codename=${distro_codename}";
};
```

Sidenote: information about the ‚origin‘ and other parameters can be found in the files /var/lib/apt/lists/*_InRelease.

Block many sites (adservers)

If you want to have a much more extensive ad-blocking done by dnsmasq one could just download the latest quite complete adserver list from pgl.yoyo.org/adservers/ (formated to be used with dnsmasq as plain text file) and save it as e.g. dnsmasq.adservers.conf in the folder /etc/dnsmasq.d/.

This can be even automated with a little script like the following – it must be run with root priviliges (e.g. with sudo) as it needs to install the new adblocking file in /etc:

```
#!/bin/bash

if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root (use sudo)"
   exit 1
fi

curl -o adservers 'https://pgl.yoyo.org/adservers/serverlist.php?hostformat=dnsmasq&mimetype=plaintext'

# Do some work on the file:
# Now remove MS-DOS carriage returns, replace 127.0.0.1 with 0.0.0.0 (much faster),
# clean up trailing blanks,
# Pass all this through sort with the unique flag to remove duplicates and save the result

sed -e 's/\r//' -e 's/127\.0\.0\.1/0\.0\.0\.0/' -e 's/[\t]*$//' < adservers | sort -u > /etc/dnsmasq.d/dnsmasq.new-
```

```
adservers.conf

rm adservers

# Seems like the dnsmasq configtest doesn't work (always returns success)
if ! /usr/sbin/dnsmasq --test; then
    printf '%s\n' 'dnsmasq configtest failed!' >&2
    rm /etc/dnsmasq.d/dnsmasq.new-adservers.conf
    exit 1
fi

mv   --force   /etc/dnsmasq.d/dnsmasq.new-adservers.conf /etc/dnsmasq.d/dnsmasq.adservers.conf
/bin/systemctl reload dnsmasq.service
```

Save this file in the /root folder (home folder for root) and add it as a cron-job for root:

```
sudo crontab -e
```

adding at the end something similar to:

```
15 04 * * * /root/adservers.sh > /root/adservers.log 2>&1
```

to run it each night at 4:15. The only thing which doesn’t seem to work is to check the dnsmasq config to be valid (the check always returns a successful check).

Keep in mind though that is only blocking DNS lookups, not IP addresses. Any direct connection to an IP address is not blocked with the above (that requires a blocking within the netfilter part; ipset is the command to look for to achieve something on the IP level).

Remote Access (VPN) via

WireGuard

```
sudo mkdir /etc/wireguard
cd /etc/wirguard
```

Configure the server

First create the keys:

```
umask 077
wg genkey > private.key
wg pubkey < private.key > public.key
```

Add a new interface for WireGuard and assign an IP address to it:

```
sudo ip link add dev wg0 type wireguard
sudo ip address add dev wg0 10.80.0.1/24
```

Configure the new interface:

```
sudo nano /etc/wireguard/wg0.conf
```

and add something along the lines of:

```
[Interface]
Address = 10.80.0.1/24
ListenPort = 40042
PrivateKey = xxxxx

[Peer]
# Smartphone
PublicKey = PEER_PUBLIC_KEY
AllowedIPs = 10.80.0.20/32

[Peer]
# Laptop
PublicKey = PEER_PUBLIC_KEY
AllowedIPs = 10.80.0.24/32
```

and finally enable it:

```
sudo ip link set up dev wg0
```

To check the status you can use the following commands:

```
sudo ip addr show dev wg0
sudo wg showconf wg0
sudo networkctl status wg0
```

To permanently enable the wireguard server (e.g. on reboot) run the following command on the shell:

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

Configure the client (e.g. smartphone)

Probably the easiest way to configure your wireguard app on the smartphone is to use a QR code – just generate it on the server command line with:

```
qrencode -t ansiutf8 < client.conf
```

To manually install the WireGuard VPN (e.g. on a laptop) create a config file:

```
nano /etc/wireguard/wg0.conf
```

with something along the lines of

```
[Interface]
PrivateKey = <LOCAL-PRIVATE-KEY>
Address = 10.80.0.24/32
DNS = 10.80.0.1

[Peer]
PublicKey = <SERVER-PUBLIC-KEY>
Endpoint = <HOST-NAME>:40042
AllowedIPs = 0.0.0.0/0, ::/0
```

To start and stop the VPN use:

```
sudo wg-quick down wg0
sudo wg-quick up wg0
```

If there are issues also don’t forget to check that traffic forwarding is allowed by linux and the firewall and that the wireguard traffic is allowed by the firewall (cf. above).

Raspberry Pi as Server

The Raspberry Pi is the great device to set up a small server at home to be used for almost any purpose. It’s cheap, small, has a low power consumption footprint and there is a great community for it to get help from. There are tons of things to do with such a mini-server. However, the very first step always will be to get the system up and running.

The best Linux to install on the Raspberry Pi is probably Raspbian which is a Debian flavor (effectively, just Debian adopted for the Raspberry Pi). Raspbian is the most stable and best supported operating system for the Raspberry with lots of helpful support. The Raspbian Linux comes in two flavors – a full system with almost everything included and a reduced, ‚lite‘ version without the graphical desktop which is the perfect choice to run the Raspberry as a server. Below is a description how to bring up the Raspberry with Raspbian-Lite and do some fine-tuning so it is a solid basis to use it as a server.

Install Linux (Raspbian-Lite)

Download the *Raspbian Lite* zip package to your computer from www.raspberrypi.org/downloads/raspbian/. You should ensure the integrity of the download by checking the SHA-256 checksum

provided on the download page. For Linux you can use the sha256sum shell command (typcially already installed and available). A useful open source tool for Windows can be found here: github.com/gurnec/HashCheck.

Best solution is to use a SSD-drive (connected to the Raspberry via USB) but alternatively a SD-card can be used. If you use a SD-card, you probably first must format the card: it must be FAT, not exFAT. Next, flash the SSD-drive or SD-card with the downloaded linux zip-package using *etcher* which is available from etcher.io (for Linux and Windows). There is no need to unzip the Rasbian file to burn the image (i.e. linux). After Etcher is finished and no errors are reported don't remove the SD card yet! The card should be mounted autmatically as well.

Copy an empty file simply called „*ssh*" (no file extension like .txt!) into the boot partition of the drive or card. Note: there should be two partitions, the other one is called „*rootfs*„, the boot partition is the one where the file „*kernel.img*" is. This will enable remote ssh access to the Raspberry right on the first boot.

If you want the Raspberry to connect to an existing WLAN right on first boot, create a file called „*wpa_supplicant.conf*„ in the boot partition. On first boot Raspbian will use this file to configure and connect to the WLAN. The file itself should contain at least the following information on the WLAN (and change the country-code to your needs):

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=us

network={
    scan_ssid=1
    ssid="My-WLAN-Name"
    psk="My-WLAN-Password"
}
```

Once done, connect the drive (or insert the SD-card) to the Raspberry. If WLAN is not used also connect it to the router (your internet gateway) with a LAN cable and finally power on the Raspberry.

Now you need to find out the IP address of the Raspberry (this very much depends on the internet router used; it might also help to assign a static IP to the Raspberry). Then connect to the Raspberry via ssh (e.g. using *putty* on Windows or ssh on Linux) and confirm the new certificate that you're asked for. The username is „*pi*" (so use „*ssh pi@raspberrypi*„)and the initial password is „*raspberry*„. If the password doesn't work, try „*raspberrz*" (due to a wrong keyboard layout) and ensure you're on the same LAN as the Raspberry.

Once logged in on the terminal (command line) the first thing is to update the whole system – for this just type in:

```
sudo apt update
sudo apt upgrade -y
```

A reboot might be required. The next step is to change the basic configuration of the server via the special config-tool for the Raspberry. To start it, simply enter and confirm with the password:

```
sudo raspi-config
```

Change the following things within this configuration application:

- Change password for the default user (option 1)
- Change hostname (option 2, N1)
- Enable old, standard interface names (option 2, N3 – select „No"!)
- Change localisation (option 4) – Locale: space and tab to select (UTF-8 preferred), select your timezone, and your wifi-country.
- Expand filesystem (option 7, A1)
- Change GPU memory to the minimum 16 (option 7, A3)

Upon „Finish“ you’re asked to reboot; do so and then log into the Raspberry once again (as user „pi“ with the new password just set).

Now you have a running system you can start using already. All the following steps are only optional and might or might not be helpful depending on what you want to achieve.

Some Improvements and Hardening

Change user to ‚admin‘

Change pi user to admin (not required, but much nicer that way). First create a new user (probably use your nickname and replace <username> with it):

```
sudo adduser <username>
sudo usermod --append --groups sudo <username>
```

You need to provide the password for that new user; all the rest being asked for can be left blank (just press Enter). Logout and login as the new user just created, then:

```
sudo usermod --login admin --home /home/admin --move-home pi
sudo groupmod --new-name admin pi
```

Logout and login as admin.

New ssh keys

Remove the standard ssh keys on the new Raspberry Pi server and generate new ones:

```
sudo rm /etc/ssh/ssh_host*
sudo dpkg-reconfigure openssh-server
```

Logout and login (confirming the new certificate once).

SSH login without password

You can use keys instead of entering a password each time you log into the new server. If you're coming from a Linux machine, then simply enter the following command on that other Linux device (if this is the first time you do this, then you might first need to generate the keys by entering ssh-keygen and simply press enter on the questions):

```
ssh-copy-id admin@my-new-server
```

Confirm once with the password – afterwards you will log into the new server without being asked for a password anymore (you identify yourself with the key on the machine your coming from).

You probably should repeat the above for other users on the same maching (e.g. the one created earlier on).

You can even improve the ssh login process even more by creating a configuration file for ssh on the server that you're coming from:

```
nano ~/.ssh/config
```

and enter something like the following (adjust to your needs):

```
Host my-server1 my-server1.mydomain
    Hostname my-server1.mydomain
    User admin

Host raspberry raspberry.lan
    Hostname raspberry.lan
    User alice
```

Then you can simply type ssh raspberry and you're logged into raspberry.lan as user alice right away. This alias configuration even works with scp as well.

Automatic security updates

One should also ensure that important security updates are installed automatically, i.e. unattended – this can be achieved with the following tool:

sudo apt install -y unattended-upgrades

and configure and enable it with (just confirm what is suggested):

sudo dpkg-reconfigure unattended-upgrades

Any updates are logged in the file /var/log/unattended-upgrades/unattended-upgrades.log.

Disable WLAN and/or Bluetooth

Most often the Bluetooth interface of the Raspberry Pi will not be used at all. Therefore, it’s a good security measure to simply disable it completely (just remember this):

sudo rfkill block bluetooth

and check it was sucessfull with:

sudo rfkill list

The same can be done with WLAN – if you’re setup and use case doesn’t require the WLAN at all just use the following:

sudo rfkill block wlan

and check the status like for Bluetooth above.

Welcome Text (motd)

If you don’t like the welcome text shown each time you log in you can easily change it by editing the motd file:

sudo nano /etc/motd

Shell behaviour

Something useful and a nice-to-have are a few shortcuts for shell commands often used. This can be easily achieved by

nano .bash_aliases

and then define aliases as wanted along these examples

```
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -lF'
alias lla='ls -AlF'

alias check='curl https://check.torproject.org/api/ip ; echo'
```

To enable them right away just run (alternatively, one could just log out and back in):

source .bash_aliases

Again, you might probably want to repeat the above for any other user on that machine.

Last but not least it proved quite useful to change the prompt so it looks different for each server (this helps mixing up different machines). You can change the prompt in the file .bashrc; look for the variable PS1 and change it to your needs (there’s enough detailed information on the internet about the details).

Configure the nano editor

The way the nano editior is working can be changed (and improved) in its config file:

nano ~/.nanorc

with e.g. the follwing configs:

```
# Use smooth scrolling by default.
```

```
set smooth

# Use the blank line below the titlebar as extra editing space.
set morespace

# Use a tab size of number columns. The value of number must be greater than 0. The default value is 8.
set tabsize 4

# Specify the color combination to use for the titlebar. Valid color names for are:
# white, black, red, blue, green, yellow, magenta, and cyan. And either "fgcolor" or ",bgcolor" may be left out.
#set titlecolor fgcolor,bgcolor

# Enable the use of ~/.nano/search_history for saving and reading search/replace strings.
unset historylog

# Save a file by default in Unix format. This overrides nano's default behavior of saving a file in the format
# that it had.  (This option has no effect when you also use set noconvert.)
set unix

# Enable mouse support, if available for your system. When enabled, mouse clicks can be used to place the cursor,
# set the mark with a double click and execute shortcuts. The mouse will work in the X Window System, and on the
# console when gpm is running. Text can still be selected through dragging by holding down the Shift key.
set mouse
```

As with some other configurations you might want to repeat this for any other users on that device.

Stop the avahi service

The avahi service is only needed in case you want to connect to e.g. a printer or scanner etc. (a device relying on

zeroconfig). On a headless server this is usually not needed and just causes unnecessary overhead on the server and the network. To disable avahi just apply:

```
sudo systemctl stop avahi-daemon.service
sudo systemctl stop avahi-daemon.socket
sudo systemctl disable avahi-daemon.service
sudo systemctl disable avahi-daemon.socket
```

Tuning systemd

The following is experimental right now and needs further investigation – but it might resolve issues that apache fails to start since it can bind to the wlan0 address on boot:

```
sudo systemctl edit apache2.service
```

and paste in:

```
[Unit]
Wants=network-online.target
After=network-online.target

[Service]
Restart=on-failure
RestartSec=10
```

then reload the config, check that it’s there then restart and check apache (all the checking below is not required, but good to know how one can check things out; also sudo is often not really mandatory):

```
sudo systemctl daemon-reload
sudo systemctl cat apache2.service
sudo systemctl show apache2.service
sudo systemctl restart apache2.service
sudo systemctl status apache2.service
sudo systemctl list-dependencies apache2.service
```

What to do with the Raspberry Server

There are lot’s of things that could be done with the Raspberry Pi server just enabled. Just A few ideas are:

- Tor WLAN Gateway
- VPN (Calling Home when you’re travelling)
- Nextcloud server (your own cloud at home)
- Host a Tor Relay (help the Tor network by sharing some bandwidth)
- OpenMediaVault (NAS)

or simply search the internet for tons of other ideas. Have fun, and happy ’serving‘!

NextCloud over Tor (onion service)

This guide is about how to set up a nextcloud instance running on a Raspberry Pi and providing the cloud service over Tor (a hidden service on the onion-network).

The initial setup of a new Raspberry Pi is always the same and described in some detail here: https://www.spaetzle.info/raspberry-server/

Install Tor

Let’s start with installing the tor package:

```
sudo apt install tor -y
```

Save the default config file as reference and create a new one:

```
sudo mv /etc/tor/torrc /etc/tor/torrc.default
sudo nano /etc/tor/torrc
```

and past in the following:

```
Log notice file /var/log/tor/notices.log

ExitPolicy reject *:*

TransPort 127.0.0.1:9040
DNSPort   127.0.0.1:5300

AutomapHostsOnResolve 1
AutomapHostsSuffixes .onion,.exit
VirtualAddrNetworkIPv4 10.42.0.0/16

HiddenServiceDir /var/lib/tor/services/nextcloud
HiddenServicePort  80 127.0.0.1:80
HiddenServicePort 443 127.0.0.1:443
```

If your running on an SD-Card (not recommended anyhow; if possible rather use a SSD-drive instead) you should add the following line to the config above:

```
AvoidDiskWrites 1
```

A crucial step is to manually create the directory for the hidden service:

```
sudo -u debian-tor mkdir /var/lib/tor/services/
```

After changing the config one should check the config, then restart the tor service and check the log file for warnings and errors:

```
sudo -u debian-tor tor --verify-config
sudo systemctl restart tor
cat /var/log/tor/notices.log
```

Firewall (nftables)

First install the firewall frontend and enable the firewall:

```
sudo apt install nftables -y
sudo systemctl enable nftables.service
```

Enable the following firewall rules, starting with a config file in your home directory

```
nano ~/nftables.conf
```

and paste in

```
#!/usr/sbin/nft -f

flush ruleset

table ip filter {
    chain input {
        type filter hook input priority 0; policy drop;

        iifname lo accept

        ct state established,related accept
        ct state invalid drop

        tcp dport ssh ct state new limit rate 10/minute accept
        tcp dport { http, https } ct state new accept

        icmp type echo-request limit rate 1/second accept
    }

    chain forward {
        type filter hook forward priority 0; policy drop;
    }

    chain output {
        type filter hook output priority 0; policy drop;
        oifname lo accept

        ct state established,related accept
```

```
        ct state invalid drop

        skuid "debian-tor" accept

        oifname eth0 udp dport ntp counter accept
        ip daddr 127.0.0.1 counter accept   # not needed ???
          ip daddr { 192.168.178.0/24, 192.168.200.0/24,
255.255.255.255 } accept
    }
}

table ip nat {
    chain input {
        type nat hook input priority 100; policy accept;
    }

    chain output {
        type nat hook output priority -100; policy accept;

        skuid "debian-tor" accept

        udp dport domain counter redirect to :5300
        ip daddr { 192.168.178.0/24, 192.168.200.0/24 } accept
         tcp flags & (fin | syn | rst | ack) == syn counter
redirect to :9040
    }
}
```

and activate these firewall rules with

```
sudo nft -f nftables.conf
```

In case something goes horribly wrong (e.g. you lock ssh sessions) you can hard reboot the server and will start without the firewall rules.

Note that nft uses its own matching of service names to port numbers – to see the list simply type in:

```
nft describe tcp dport
```

Once you’re happy with them working make them permanent with

copying them to the standard place (enabled on reboot):

```
sudo cp /etc/nftables.conf /etc/nftables.conf.default
sudo cp nftables.conf /etc/nftables.conf
```

Install Nextcloud

Install php

Start by installing php with:

```
sudo apt install -y apache2 mariadb-server libapache2-mod-php php-gd php-json php-mysql php-curl php-mbstring php-intl php-imagick php-xml php-zip php-apcu
```

Prepare MySQL (MariaDB)

To initialize the MariaDB database start with:

```
sudo mysql_secure_installation
```

and answer the questions accordingly (e.g. remove anonymous user). Now the database is ready and we create a nextcloud-user in mysql: Log into MariaDB database server with the following command:

```
sudo mysql -u root
```

Then create a database for NextCcoud using the MariaDB command below. This name of the database could be nextcloud (but one can use whatever name is prefered). Note: Don’t leave out the semicolon at the end.

```
> create database nextcloud;
```

Then create a new user. Again, you can use your preferred name for this user. Replace ‚your-password‘ with your preferred password (leave the single quotes in place):

```
> grant all privileges on nextcloud.* to nextclouduser@localhost identified by 'your-password';
```

The above command will create the user and grant all privileges. Now flush MariaDB privileges and exit:

```
> flush privileges;
> exit;
```

Install Nextcloud package

To download the files, first get the download link in a browser (on nextcloud.com, download section, server packages), copy the link and then use the wget command (note that the actual filename will change once new versions of nextcloud will be released):

```
wget
https://download.nextcloud.com/server/releases/nextcloud-x.y.z
.zip
```

and download the checksum (just add „.sha256“ to the above download command):

```
wget
https://download.nextcloud.com/server/releases/nextcloud-x.y.z
.zip.sha256
```

and check it with:

```
sha256sum -c nextcloud-x.y.z.zip.sha256
```

and then unzip the downloaded nextcloud package, copy it to the webserver directory and change the ownership:

```
unzip nextcloud-x.y.z.zip
cp -r nextcloud /var/www
sudo chown -R www-data:www-data /var/www/nextcloud/
```

Enable the apache webserver

First, lets tell apache to list on which IP addresses and which ports:

```
sudo nano /etc/apache2/ports.conf
```

and fill it with something along (but change to your local IP addresses):

```
Listen 127.0.0.1:80 http
Listen 192.168.200.42:80 http


<IfModule ssl_module>
        Listen 127.0.0.1:443 https
        Listen 192.168.200.42:443 https
</IfModule>

<IfModule mod_gnutls.c>
        Listen 127.0.0.1:443 https
        Listen 192.168.200.42:443 https
</IfModule>
```

Next we create a config file for our actual nextcloud instance

```
sudo nano /etc/apache2/sites-available/nextcloud.conf
```

and paste in:

```
ServerName abc.mynet

<VirtualHost 127.0.0.1 192.168.200.22>
        ServerName abc.mynet
                                                ServerAlias
h72qy8dg3rhd55rn7u3zkaibw4598dupq544wrlqsmx4d3oxjxvuurad.onion
        DocumentRoot /var/www/nextcloud/
</VirtualHost>


<Directory /var/www/nextcloud/>
  Options +FollowSymlinks
  AllowOverride All

 <IfModule mod_dav.c>
  Dav off
 </IfModule>

 SetEnv HOME /var/www/nextcloud
```

```
 SetEnv HTTP_HOME /var/www/nextcloud

</Directory>
```

To let apache check the config for errors use:

```
sudo apache2ctl configtest
```

Finally, enable this new config together with two required apache modules:

```
sudo a2ensite nextcloud.conf
sudo a2dissite 000-default.conf
sudo a2enmod rewrite
sudo a2enmod headers
sudo a2dismod status
```

Before actually activating the new config we apply a few more things. First some additional measures to improve anonymity:

```
sudo nano /etc/apache2/conf-enabled/security.conf
```

and change it so it shows these two configs:

```
ServerTokens Prod
ServerSignature Off
```

Finally activate all changes by restarting apache:

```
sudo systemctl reload apache2
```

Fire up nextcloud

Configuration

To connect to the database just point your webbrowser to your new nextcloud server and complete the installation wizard. This also creates the basic config file for nextcloud which we also need to change manually a bit:

```
sudo nano /var/www/nextcloud/config/config.php
```

One should add additional so-called trusted domains; here we

want to add out onion web-address. To get your new onion address look it up here:

sudo cat /var/lib/tor/services/nextcloud/hostname

so with a few other addional tweaks, part of your config file (not a complete example!) might look like:

```
  'trusted_domains' =>
  array (
    0 => 'localhost',
    1 => '127.0.0.1',
    2 => '192.168.202.44',
    3 => 'xxx.bet',
                                                4            =>
'h9dfype6yrhd55rn7u3dk7ebwhhkgospq544wrlqsmx4d3oxjxvuur99.onio
n',
  ),
  'overwrite.cli.url' => 'http://xxx.bet',
  'memcache.local' => '\OC\Memcache\APCu',
'htaccess.RewriteBase' => '/',
  'trashbin_retention_obligation' => 'auto,90',
```

Php configuration

The php config should be changed to e.g. accept uploads of larger files (note that the php version number might be different):

sudo nano /etc/php/7.3/apache2/php.ini

and change (search for the options in this very lengthy config file):

```
memory_limit = 512M
post_max_size = 256M
upload_max_filesize = 256M
```

crontab

You might improve a bit on the nextcloud performance by using cron:

sudo crontab -u www-data -e

and add at the very bottom:

*/15 * * * * /usr/bin/php -f /var/www/nextcloud/cron.php

Finally, log into nextcloud and on the admin panel enable cron.

Update Nextcloud

Although there is a possibility to update your Nextcloud instance via the web frontend this might be failing in same cases due to time-outs. The safer approach is to simply run:

```
cd /var/www/nextcloud/updater
sudo -u www-data php ./updater.phar
```

on the command line interface of your machine.

Host a Tor Relay

If you want to support the Tor project and you have some bandwidth to share (at least 10 Mbps in both directions, i.e. download and upload) you should consider hosting a Tor Relay. It can be done quite easily with a Raspberry Pi or any similar hardware as described below.

Our assumption is that we connect the Raspberry Pi with an ethernet cable to our ISP router and that we have a wireless LAN running for our private, local machines already. We will connect the Raspberry Pi to this WLAN for administration purposes (so this is not done over the internet connection on eth0).

Initial Setup as a Server

The very first step to start out is to get a new Raspberry Pi and do the basic setup as a headless server (described in a separate post). For the sake of increased security you shouldn’t use it for anything else than just to run the Tor Relay, so the Raspberry Pi should be dedicated to this task.

Install Tor and configure it

First we add the official repository of the tor-project for Debian (as outlined at www.torproject.org/docs/debian.html.en):

```
sudo nano /etc/apt/sources.list.d/torproject.org.list
```

and put in the following:

```
deb https://deb.torproject.org/torproject.org buster main
```

Note that you might need to replace the debian release name which is currently „buster“ with a newer one.

Then install the keys signing the tor packages (maybe good to check torproject.org/docs/debian.html for the latest updates):

```
sudo gpg --recv A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89
sudo gpg --export A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89 | sudo apt-key add -
```

and finally install the tor software (including the recommended package to keep the key updated):

```
sudo apt update
sudo apt install -y tor deb.torproject.org-keyring
```

To configure the tor service as a relay edit the config file:

```
sudo mv /etc/tor/torrc /etc/tor/torrc.default
sudo nano /etc/tor/torrc
```

and enter something like (change the details to your setup as required):

```
NickName MyNewRelay
ContactInfo myemail@example.org

Log notice file /var/log/tor/tor.log
AvoidDiskWrites 1

ExitRelay 0
ExitPolicy reject *:*

ORPort  6080 IPv4Only
DirPort 6443 IPv4Only

RelayBandwidthRate  10 Mbit
RelayBandwidthBurst 15 Mbit

TransPort 127.0.0.1:9040
DNSPort   127.0.0.1:5300

AutomapHostsOnResolve 1
AutomapHostsSuffixes .onion,.exit
VirtualAddrNetworkIPv4 10.10.0.0/16
```

Now let tor check if the above config file is correct:

```
sudo -u debian-tor tor --verify-config
```

The last line should say that everything looks fine.

The contact information is optional but might be quite handy for others if there should be something strange with the relay. It doesn’t have to be an email address but could be any kind of text.

In case you are running more than just one Tor Relay you have to also include a „MyFamily“ option in the config above and list all your key-fingerprints of your Tor Relays in each of the torrc config files. You get the fingerprint with sudo -u debian-tor tor –list-fingerprint and remember that there must

be a $ (Dollar-sign) at the beginning of each fingerprint.

Port Forwarding on ISP Router

In our example the Raspberry Pi (our Tor Relay) sits behind a router which is the gateway into the internet (often provided by the ISP). With the tor configuration above we need to establish port forwarding on this internet router, so TCP traffic coming from the internet (on ports 6080 and 6443) is forwarded to out Tor Relay (on the same ports).

If you would like to use other ports to the outside world (internet) than on the Tor Relay server itself, the Tor config file (torrc) needs to have something like:

```
ORPort    80 NoListen
ORPort  6080 NoAdvertise
DirPort  443 NoListen
DirPort 6443 NoAdvertise
```

The port forwarding on the ISP router then obviously has to forward ports 80 and 443 to ports 6080 and 6443 respectively on the Tor Relay.

The ports chosen are kind of arbitrary and we are free to take whatever we like. One advantage of advertising (i.e. using) ports 80 and 443 towards the internet is that they are very unlikely to be blocked as they are usually taken for http and https traffic. The drawback is that you can’t use these ports for something else (like a web-presence). Also some routers seem to have issues with port-forwarding these ports (e.g. lost after a router-reboot).

The details on how port forwarding is configured on the internet router depends heavily on that device but usually each of these kind of routers offers this feature somehow (just search the internet in case this is not obvious).

Start and Test it!

First restart Tor (so it picks up the latest configuration):

```
sudo systemctl restart tor
```

Check the logs for what Tor does and if it complains about anything – the following commands might be useful to check for any errors:

```
sudo systemctl status tor.service
sudo cat /var/log/tor/tor.log
journalctl | grep Tor
```

You are perfectly fine if you see something like „*Self-testing indicates your ORPort / DirPort is reachable from the outside*„. If there are no issues your new Tor Relay will also become visible on the torproject metrics-webpage at metrics.torproject.org/rs.html (this might take a few hours though, so be patient).

One could also increase the level of logging information written by tor. Just change the option in the /etc/tor/torrc configuration file – after the „log“ statement one could place either debug, info, notice, warn, or err. Additionally, one could (temporarily, for debugging) turn off the scrubbing of sensitive information in the log-files as well. So for debugging include something like the following in the torrc

```
SafeLogging 0
Log info file /var/log/tor/tor.log
```

Once running fine one should probably keep the logging at the ’notice‘ level though.

Also note that it takes up to 2 months until a new Tor Relay gets fully used – and since there is not always traffic available it will mostly never really run at the full possible bandwidth. See this article for some background on it: blog.torproject.org/lifecycle-new-relay.

Enable the Firewall (nftables)

First install the firewall frontend and enable the firewall:

```
sudo apt install nftables -y
sudo systemctl enable nftables.service
```

Enable the following firewall rules, starting with a config file in your home directory

```
sudo nano ~/nftables.conf
```

and paste in (ensure the port numbers of tor defined above match the firewall rules here!):

```
#!/usr/sbin/nft -f

flush ruleset

table ip filter {
    chain input {
        type filter hook input priority 0; policy drop;

        iifname lo accept

        ct state established,related accept
        ct state invalid drop

        tcp dport ssh ct state new limit rate 10/minute accept

        iifname eth0 tcp dport {6080, 6443} accept

        icmp type echo-request limit rate 1/second accept
    }

    chain forward {
        type filter hook forward priority 0; policy drop;
    }
```

```
    chain output {
        type filter hook output priority 0; policy drop;
        oifname lo accept

        ct state established,related accept
        ct state invalid drop

        skuid "debian-tor" accept

        oifname eth0 udp dport ntp counter accept
         oifname wlan0 tcp dport 8086 accept     # sending
measurements to influxdb
        oifname wlan0 udp dport domain counter accept   # not
needed ???
        ip daddr 127.0.0.1 counter accept       # not needed
???
          ip daddr { 192.168.178.0/24, 192.168.200.0/24,
255.255.255.255 } accept
    }
}

table ip nat {
    chain input {
        type nat hook input priority 100; policy accept;
    }

    chain output {
        type nat hook output priority -100; policy accept;

        skuid "debian-tor" accept

        ip daddr { 192.168.178.0/24, 192.168.200.0/24 } accept
         tcp flags & (fin | syn | rst | ack) == syn counter
redirect to :9040
#        udp dport domain counter redirect to :5300
    }
}
```

and then activate these firewall rules with

```
sudo nft -f nftables.conf
```

One can check the applied firewall rules (and also see the counters) with

sudo nft list ruleset

In case something goes horribly wrong (e.g. you locked your own ssh session) you can hard reboot the server and will start without the firewall rules.

Note that nft uses its own matching of service names to port numbers – to see the list simply type in:

nft describe tcp dport

Once you're happy with them working make them permanent with copying them to the standard place (enabled on reboot):

```
sudo cp /etc/nftables.conf /etc/nftables.conf.default
sudo cp nftables.conf /etc/nftables.conf
```

Setup dnsmasq

We use dnsmasq to handle all DNS requests initially since there we can configure which upstream server to use for which domains. In our case we use the (local, private) DNS server 192.168.200.1 on wlan0 to resolve our private domain and send everything else into Tor.

Install dnsmasq, save its default configuration for later reference and edit the config:

```
sudo apt install -y dnsmasq
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.default
sudo nano /etc/dnsmasq.d/dnsmasq.conf
```

and paste in:

```
# Configuration file for dnsmasq (DNS part, DHCP is disabled by default)

#Optional logging can be enabled to support problem
```

```
determination
#log-dhcp
#log-queries
#log-facility=/var/log/dnsmasq.log

# Listen on these IP addresses for DNS requests (always
include localhost):
listen-address=127.0.0.1

# Where to send DNS request to which can't be resolved
locally.
# Here we use the local Tor service listening on port 5300
(see /etc/tor/torrc)
server=127.0.0.1#5300
server=/alpha/192.168.200.1

# The bind-interfaces directive instructs dnsmasq to bind only
to
# the network interface specified in the listen-address
directive.
bind-interfaces

# The no-hosts directive instructs dnsmasq not to read
hostnames
# from /etc/hosts.
no-hosts

# Read hostname information from this file in addition
addn-hosts=/etc/dnsmasq.hosts

# Never forward plain names (without a dot or domain part)
upstream
domain-needed

# Never forward addresses in the non-routed address spaces.
bogus-priv

# If you don't want dnsmasq to read /etc/resolv.conf or any
other
# file, getting its servers from this file instead, then
uncomment this.
```

```
no-resolv

# If you don't want dnsmasq to poll /etc/resolv.conf or other
resolv
# files for changes and re-read them then uncomment this.
no-poll

# Set this if you want to have a domain
# automatically added to simple names in a hosts-file.
expand-hosts

# Number of hostnames being cached in RAM for DNS lookups:
cache-size=10000
```

and save the file. To test if the config for dnsmasq is free of errors check the syntax with:

```
sudo dnsmasq --test
```

We also need to disable that dnsmasq takes nameservers from resolvconf – edit the file

```
sudo nano /etc/default/dnsmasq
```

and uncomment the last line so it reads:

```
...
IGNORE_RESOLVCONF=yes
```

Now start the dnsmasq service:

```
sudo systemctl start dnsmasq.service
sudo systemctl status dnsmasq.service
sudo systemctl enable dnsmasq.service
```

Finally do a few tests (abc is fine, server1 must be a valid hostname on your local network):

```
host -v cnn.com
host -v abc.onion
host -v server1.alpha
```

Some Improvement

Change Tor Repository

We can change the repository for the tor packages to their onion server:

sudo nano /etc/apt/sources.list.d/torproject.org.list

and change it to:

```
# Repository for the Tor packages:
# deb https://deb.torproject.org/torproject.org buster main

deb http://sdscoq7snqtznauu.onion/torproject.org buster main
```

However, Debian is not supporting anonymous and safe software repos by standard, so we need to enable it first – create a new apt config file:

sudo nano /etc/apt/apt.conf.d/80onion-repos

and put in just one line:

```
Acquire::BlockDotOnion "false";
```

Now finally update and upgrade the system with:

```
sudo apt update
sudo apt upgrade -y
```

Automatic updates for Tor

It’s also recommended to get the tor software upgraded automatically. Assuming that the package ‚unattended-upgrades‘ is installed already, we simply need to add the updates from the torproject to the config:

sudo nano /etc/apt/apt.conf.d/50unattended-upgrades

and add the line with the TorProject repository:

```
...
     "origin=Debian,codename=${distro_codename},label=Debian-
Security";
    "origin=TorProject,codename=${distro_codename}";
...
```

Sidenote: information about the ‚origin‘ and other parameters can be found in the files /var/lib/apt/lists/*_InRelease.

Monitoring

It’s good practice to regularly check the log files for errors for anything unusual:

```
sudo tail -25 /var/log/tor/tor.log
sudo tail -f /var/log/kern.log
```

Also check the torproject metrics-webpage at metrics.torproject.org/rs.html from time to time which should give a sufficient overview of the Tor Relay status. If you can browse onion websites you could even use the onion-metrics site at: rougmnvswfsmd4dq.onion/rs.html.

In case a more in-depth monitoring is required have a look at e.g. vnstat, theonionbox or (most advanced, usually not really required and with lots of shortcommings) Telegraf with InfluxDB and Grafana.

Backup using rsync

Many people tend to ignore how important it is to have regular backups of their data until something bad happens and their stuff is gone for good. The good news is that on Linux there is an easy way to automatically create regular backups (it should work on other systems like Windows as well with some

tweaking). One can even keep some of the backups for a very long time which might come in handy if you recognize something was lost like months ago. The obvious option is to use the rsync program as basis for a remote backup system.

The rsync tool

Something very smart with rsync it that one can point it to a previous, already existing backup on the server (cf. the –link-dest option in the script below) and rsync will compare any file of the new backup to the data there. If a file is existing in the old backup already rsync will not transfer the file again, but simply hard-link to it on the server and therefore also (almost) not consume any additional storage.

Another advantage of rsync comes into play with huge files (think of videos or veracrypt containers): rsync compares a file to a previous version on the server on a block level and only transfers and updates the parts of the file that did change.

Prepare the storage

First of all to make backups you need some kind of external storage. Theoretically, you could make backups on the same machine but if that one’s lost, stolen or broken all is gone (it might be useful though to go back in time for data which was deleted or overwritten by mistake). The next best option is to have some external storage stick or drive which is manually connect to your computer to run manual backups from time to time.

The best option for normal (i.e. non-business critical) usage seems to be a small storage server (NAS) on the local network. A good and rather cheap option is for example a Raspberry Pi running OpenMediaVault connected with an external SSD.

Enable secure access (ssh)

The very first step is to enable the ssh service as such on the storage server (e.g. OpenMediaVault). How this is done exactly depends on the server but usually this should be installed and enabled by default. Otherwise just search around or there should be some documentation on the internet.

For an automated, unattended backup to another server we need to enabled ssh access based on cryptographical keys without the need to an interactive password.

The first step is to create the required keys on the device which will be backed up (your laptop for example):

```
ssh-keygen -t rsa
```

Answer all questions with return, i.e. keeping the defaults and don’t enter a password!

Next, we need to copy the public key which was just created from the device to the remote server where the backups will be stored (take your username and the name of your server) and enter your remote password when asked for it:

```
ssh-copy-id user@hostname
```

The private, secret key will always stay on your local machine; typically it will be stored in the directory .ssh.

Now give it a try to see if everything works out fine – just enter (again take your own user name and server name):

```
ssh user@hostname
```

You should be connected to your remote shell right away without any password requests. In case it shouldn’t work try the command with ssh -v or ssh -vv and check the output for an indication what might be wrong.

To make life a bit easier you might also want to configure

some of your ssh connections so they will be easier to use. To do so, just open the config file with:

```
nano ~/.ssh/config
```

and enter your details as needed like below:

```
Host mediaserver mediaserver.alpha
    Hostname mediaserver.alpha
    User admin

Host storage
    Hostname storage.alpha
    User tom
```

There are tons or options that can be defined in this ssh config file – start out with man ssh_config to check them out.

Rsync backup (push to server)

You need to decide where to store the backup script on your device (typically a laptop nowadays) and a good idea is to create a dedicated directory in your home directory. It’s also wise to make it hidden, i.e. to start the name with a dot so it won’t be visible by standard and it won’t be backed up by the script.

```
mkdir ~/.rsync
cd ~/.rsync
nano backup.sh
```

and paste in the following while changing everything specific for your local environment and setup (e.g. path names, user name, maximum bandwidth, etc.):

```
#!/bin/bash

LOGFILE="${0%.*}".log

# To ensure only one instance of the backup-script is running
```

```
we create a lock-dir first.
# The lock is removed automatically on exit (including
signals).
if ! mkdir "${0%.*}".lock; then
     echo "Lock detected (either rsync still running or
manually locked); backup aborted..."
    exit 1
fi
trap 'rm --recursive --force --one-file-system "${0%.*}".lock'
EXIT

# In case anything is failing during execution we want to
catch it here and stop the script.
# Unfortunately, the following doesn't work for commands
within a 'if' query... !
trap 'echo "Error encounted while executing $BASH_COMMAND.
Exiting..." >> $LOGFILE; exit 1' ERR

echo "Starting new backup..." $(date) > $LOGFILE

SOURCE="/home/tom/"
SERVER="tom@storage.alpha"
BACKUP="/mnt/backup/laptop/tom/latest"
TARGET="$SERVER:$BACKUP"

if ssh $SERVER "test -e '$BACKUP'"; then
    echo "Latest full backup still exists (not archived yet).
Exiting..." >> $LOGFILE
    exit 1
fi

RSYNCOPTIONS="--archive --numeric-ids --one-file-system --
exclude-from=.rsync/backup.exclude --link-dest=../hourly.0 --
compress --bwlimit=400K --partial-dir=.rsync-partial --human-
readable --stats"

ionice -c2 -n7 nice -n 19 rsync $RSYNCOPTIONS "$SOURCE"
"$TARGET.tmp" >> $LOGFILE

ssh $SERVER "mv '$BACKUP.tmp' '$BACKUP'"         # Put the
backup in place, so it's marked as completed
```

```
ssh $SERVER "touch '$BACKUP'"                    # Timestamp
the backup

echo "Backup script completed..." $(date) >> $LOGFILE
```

and make it executable:

```
chmod u+x backup.sh
```

Last step for the backup is to create a small file called backup.exclude which contains what will not be backed up. An example could be:

```
# Always exclude files and directories with the following
endings
*.part
*.iso
*.img
*.log
*.bak
*.old
*.zip
*.7z
*.tmp
*.temp
*.core
*.lock
*.crdownload
*.mp4

# As an exception to the rule below we include the following
hidden directories
+ .rsync/
+ .hades/

# Now exclude all hidden files and directories (starting with
a dot) from the backup
.*

# Exclude temporary folders as well
*/temp/
*/tmp/
```

```
# And finally exclude any confidential files that might be mounted to this directory
Hades/
```

With these exclude filters everything (files and directories) starting with a dot in the name and all files ending on .part or .iso are excluded from the backup.

First of all you should try the backup manually; maybe with a rather small scope of files to backup (so it doesn’t run for hours while testing). To start the script, simply type

```
.rsync/backup.sh
```

in your home directory and check the log-file in the same directory and the backups on the server. Note: if the script is run by cron then the it will be run from your home directory, therefore the relative path to the exclude-file is relative to the home directory. If run from another directory the relative path to the exclude file must be changed accordingly.

Now run a full backup which easily can take many hours. Then manually rename that backup on the server from latest to hourly.0 and run the backup script again. This time it should complete within a few minutes at most. Check the new backup on the server and will find the hard-linked files there.

If everything looks fine one probably wants to run the script automatically like for example each hour. To enable this simply make in entry into crontab with:

```
crontab -e
```

and add the following line (or something similar) at the end of the file:

```
0 * * * * /home/tom/.rsync/backup.sh
```

In case you want to avoid the script to be running for some time simply create the lock manually (in the script

directory):

mkdir backup.lock

You might want to check the log-file in the same directoy to see if everything is working as it should. And don’t forget to remove the lock in case you created it manually and you want to run the backups again:

rmdir backup.lock

If you want to lock the backup script quite frequently it might be a good idea to define alias commands for the locks.

Archive of backups

Usually, one also wants to keep a few of the backups for a longer time. This means that some of the older backups should be kept for the future. This is accomplished with archiving the backups which can be easily automated with the following shell script.

Connect to your storage (NAS) server and become root to save the script:

```
su
nano /root/backup-archive.sh
```

Now paste in the following – adjusting the path $BASE and a few other things maybe to your local setup and needs (don’t worry, the script looks much more complicated than it actually is):

```
#!/bin/bash

# To ensure only one instance of the backup-script is running we create a lock-dir first.
# The lock is removed automatically on exit (including signals).
if ! mkdir "${0%.*}".lock; then
```

```
     echo "Lock detected (either script still running or
manually locked). Exiting..."
    exit 1
fi
trap 'rm --recursive --force "${0%.*}".lock' EXIT

# In case anything is failing during execution we want to
catch it here and stop the script.
# Unfortunately, the following doesn't work for commands
within a 'if' query... !
trap 'echo "Error encounted while executing $BASH_COMMAND.
Exiting..."; exit 1' ERR

BASE="/mnt/backup/laptop/tom"

N=100   # Maximum number of backups per category

case $1 in
    hourly)
        if [ ! -d "$BASE/latest" ]; then
            echo "No new backup available to be archived (no
folder 'latest'). Exiting..."
            exit
        fi
        # If the latest backup is identical to the previous
one in 'hourly.0' then skip
        # the backup rotation. Exit status of 'diff' is 0 if
inputs (directories) are
        # identical, 1 if they are different, 2 if there's any
kind of trouble.
          if diff --recursive --brief --no-dereference
$BASE/latest $BASE/hourly.0; then
            echo "Not rotating, since there are no changes in
'latest' since last backup."
            echo "Removing 'latest' so a new backup will be
made."
            rm --recursive --force "$BASE/latest"
            exit
        fi
        rm --recursive --force "$BASE/hourly.8"
        for I in {100..0}; do
```

```
                if [ -d "$BASE/hourly.$I" ]; then mv
"$BASE/hourly.$I" "$BASE/hourly.$((I+1))"; fi
        done
        mv "$BASE/latest" "$BASE/hourly.0"
        ;;
    daily)
         until [ -d "$BASE/hourly.$N" ]; do      # Keep at
least hourly.0 for the hard links
             if (( $N == 1 )); then echo "No hourly backup
available for daily backup. Exiting..."; exit; fi
            let N--
        done
        rm --recursive --force "$BASE/daily.5"
        for I in {100..0}; do
                 if [ -d "$BASE/daily.$I" ]; then mv
"$BASE/daily.$I" "$BASE/daily.$((I+1))"; fi
        done
        mv "$BASE/hourly.$N" "$BASE/daily.0"
        ;;
    weekly)
        until [ -d "$BASE/daily.$N" ]; do
             if (( $N == 0 )); then echo "No daily backup
available for weekly backup. Exiting..."; exit; fi
            let N--
        done
        rm --recursive --force "$BASE/weekly.4"
        for I in {100..0}; do
                if [ -d "$BASE/weekly.$I" ]; then mv
"$BASE/weekly.$I" "$BASE/weekly.$((I+1))"; fi
        done
        mv "$BASE/daily.$N" "$BASE/weekly.0"
        ;;
    monthly)
        until [ -d "$BASE/weekly.$N" ]; do
             if (( $N == 0 )); then echo "No weekly backup
available for monthly backup. Exiting..."; exit; fi
            let N--
        done
        rm --recursive --force "$BASE/monthly.12"
        for I in {100..0}; do
                if [ -d "$BASE/monthly.$I" ]; then mv
```

```
"$BASE/monthly.$I" "$BASE/monthly.$((I+1))"; fi
        done
        mv "$BASE/weekly.$N" "$BASE/monthly.0"
        ;;
    yearly)
        until [ -d "$BASE/monthly.$N" ]; do
            if (( $N == 0 )); then echo "No monthly backup
available for yearly backup. Exiting..."; exit; fi
            let N--
        done
        for I in {100..0}; do
                if [ -d "$BASE/yearly.$I" ]; then mv
"$BASE/yearly.$I" "$BASE/yearly.$((I+1))"; fi
        done
        mv "$BASE/monthly.$N" "$BASE/yearly.0"
        ;;
    *)
        echo "Invalid (or no) option. Exiting..."
        ;;
esac
```

It must be invoked by giving an argument (either hourly, daily, weekly, monthly, or yearly) depending on what level of backups should be archived.

Once you confirmed it’s working by running it manually a few times the best practice is to invoke it automatically and regularly by setting up crontab. The different levels of rotations should be run at different daytimes, e.g. run the yearly one at 3:10 am (once a year), the monthly one at 3:20 am (once a month) and so on. The hourly rotation should be scheduled a few minutes before the new backup on the laptop runs – e.g. run the hourly rotation at 10 mins before the top of the hour if the backup script on the source device (laptop) runs at the full hour.

Just one example for cron (setup via sudo crontab -e):

```
20 * * * * /root/backup-archive.sh hourly
25 3 * * * /root/backup-archive.sh daily
30 4 * * 1 /root/backup-archive.sh weekly
```

```
35 4 1 * * /root/backup-archive.sh monthly
40 4 1 1 * /root/backup-archive.sh yearly
```

Retrieve files

Retrieving documents from the backup is really easy. Simply use a command like the following (maybe first switch to a separate local directory first):

```
mkdir retrieve
cd retrieve
nice rsync --protect-args --archive --numeric-ids --progress tom@storage.alpha:/mnt/backup/laptop/tom/hourly.4/Lyrics/Elvis.pdf .
```

Obviously, the above rsync command needs to be adapted to the specific setup (user name, host name, path, file to retrieve, etc.) and one could also limit the bandwidth just like in the backup script.

Final thoughts

This backup setup and the two shell scripts are quite simplistic and in no way elaborated – but the whole thing just works. It’s also worth mentioning that there are applications around that effectively implement something pretty similar; one example being the ‚rsnapshot‘ tool. Personally, I prefer to do it myself though as this gives much more flexibility control, I can learn something – and it’s just plain fun to see it working.