Contents

Generate AllowedIPs Text / QR Code in Terminal For Granular WireGuard® Tunnel Exclusions


```
This project is **NOT** affiliated with WireGuard® in any way, nor is it officially supported in any way. I just
```

Credits

```
(C) 2020, copyright@mzpqnxow.com, please see licensing terms at end of document*
```

Background: WireGuard®

Hopefully you know by now, but WireGuard® is an extremely simple yet fast and modern VPN that utilizes state-of-the-art cryptography. You can read more about it here

Generate AllowedIPs Value As Text String or QR Code in Terminal For Granular WireGuard® Tunnel Exclusions

Given a list of IPv4 addresses (dotted quad, CIDR) print an inclusion list (`AllowedIPs` string) suitable for the WireGuard® mobile app, optionally generating a QR code using ANSI escape characters in a standard terminal window. Useful for excluding only a partial set of LAN (or WAN) addresses from the WireGuard® tunnel. Currently, the only "easy" option for accessing a LAN is to opt *all* RFC1918/RFC3330 private addresses out of the tunnel, using the checkbox that the WireGuard® client provides. This script allows a more granular specification, and also allows the specification of specific WAN addresses as well

Quick-Start (See Usage For Detailed Instructions)

Generate a fine-grained `AllowedIPs` value that excludes only three specific LAN addresses to bypass the tunnel:

```
$ ./generate-wireguard-allowed-qr.py --exclude 192.168.1.2 192.168.1.3 192.168.1.10 --qr
```

This command will generate a QR code which will be printed to your terminal/TTY containing the `AllowedIPs` value, and will also print a text version

Specific Use-Case (LAN + Security Concerns)

Consider a mobile user with an on-demand WireGuard® VPN connection for the purpose of both privacy and security that needs to access only a small subset of LAN hosts (or WAN hosts) outside of the tunnel. By using fine-grained addresses in the `AllowedIPs` parameter, this can be accomplished without hassle. The effect:

- As usual, the tunnel is used by default for Internet resources
- – Protects privacy from ISPs that mine and sell browsing habits
- Allow access to only *specific* LAN resources as specified by the exclusion list on the command-line
- – Protects the client from rogue LAN devices by reducing the amount of devices that can interact with your device
- – Protects the broader LAN in the event that your device is compromised by limiting the amount of LAN addresses your device can access

Natively Supported WireGuard® Options for LAN Access via Tunnel Bypass

One existing and very simple solution is to use the built-in "Exclude Private IPs" option in the WireGuard® mobile app, which generates a fixed list that simply generates an `AllowedIPs` address list that excludes all RFC1918/RFC3330 Private Addresses. However, this is a very broad range and is not trivial to tweak

Using this rinky-dink little app, you can specify granular exclusions, e.g. `192.168.1.2 192.168.1.3 192.168.1.4` for the WireGuard® tunnel and receive a list of networks suitable for the `AllowedIPs`

Allowing, e.g., only 192.168.1.2, 192.168.1.3 and 192.168.1.4 (as opposed to 192.168.0.0/16) effectively protects the rest of the 192.168.0.0/16 network from being exposed to the device. It also effectively protects the device from the rest of the 192.168.0.0/16 network (and 10/8 and 172.16/12 as well, if relevant)

Why Is Code Required? Can't I Just Specify The List Myself?

In case it isn't already clear, WireGuard® mobile only allows the client to specify IP addresses that are "allowed" to use the tunnel. It is an opt-in system, most easily used when performing a LAN-to-LAN tunnel. It does not allow the user to specify a blacklist- to do this, a long list of "non-blacklisted" networks must be generated. Because of this, you can't briefly or easily say "all traffic except to/from host 192.168.1.3 should transit the tunnel easily". Instead, you need a list of CIDR notation networks that make up the entire IPv4 address space, excluding "192.168.1.3".

If you're curious, the value for this specific example looks like this and is rather unwieldy, which is where the QR code functionality comes in handy:

```
0.0.0.0/1,128.0.0.0/2,224.0.0.0/3,208.0.0.0/4,200.0.0.0/5,196.0.0.0/6,194.0.0.0/7,193.0.0.0/8,192.0.0.0/9,192.19
```

This is a bit much to effectively specify just a single host for tunnel bypass, and is not so simple to construct in your head, unless you're an advanced human subnet calculator

Sad Things

- If/when the a `Disallowed IPs` option is provided in the WireGuard® client, to complement/provide and alternative to the `Allowed IPs` option, this script will become 100% worthless. Hopefully this happens at some point because it's annoying to generate lists like this, even with the convenience of the QR code mechanism

Caveat Emptor / NOTES

- YMMV, requires thorough testing
- You may need to explicitly include peer address for tunnel bypass!
- Feedback is welcome, this is an experimental script. See also the TODO section ...

Usage

Given a list of dotted-quad and CIDR notation IPv4 networks on the command-line, generate the inverse in the format WireGuard® prefers. Optionally, produce a QR code on the terminal screen that scan be easily scanned into mobile devices

This was written to create an "Allowed IPs" list for the WireGuard® mobile app to allow granular tunnel bypass. WireGuard® currently doesn't allow you to supply an "Excluded IPs" option, so you have to invert the networks that you want to bypass the

tunnel. This overengineered app will do that for you. One very convenient feature is printing a QR code so that the Allowed IPs list will be easy to copy to a mobile device

Help Output

```
$ ./generate-wireguard-allowed-qr.py --help
usage: ./generate-wireguard-allowed-qr.py [-h] --exclude n.n.n.n[/mask]
                                          [n.n.n.n[/mask] ...] [--loose] [-D]
                                          [-q] [-l] [-4] [-W] [-C]

optional arguments:
  -h, --help            show this help message and exit
  --exclude n.n.n.n[/mask] [n.n.n.n[/mask] ...]
                        List of networks and IP addresses separated by commas
                        for exclusion
  --loose               Specify strict CIDR block definitions, fail when the
                        host bit is set in a CIDR block
  -D, --debug           Enable debug messages
  -q, --qrencode        Output a QR code to the screen using ANSI escape chars
  -l, --lines-output    Produce output in line-based format
  -4, --no-ip6          Do not include ::0/0 output
  -W, --no-wireguard-format
                        Exclude the "AllowedIPs = " literal
  -C, --no-csv-output   Disable output of single-line CSV (WireGuard-style)
                        format
```

Examples

Use WireGuard® tunnel for all networks except 192.168.1.0/24 and 192.168.2.0/24:

```
$ ./generate-wireguard-allowed-qr.py --exclude 192.168.1.0/24 192.168.2.0/24
AllowedIPs = 0.0.0.0/1,128.0.0.0/2,192.0.0.0/9,192.128.0.0/11,192.160.0.0/13,192.168.0.0/24,192.168.3.0/24,
192.168.4.0/22,192.168.8.0/21,192.168.16.0/20,192.168.32.0/19,192.168.64.0/18,192.168.128.0/17,192.169.0.0/16,
192.170.0.0/15,192.172.0.0/14,192.176.0.0/12,192.192.0.0/10,193.0.0.0/8,194.0.0.0/7,196.0.0.0/6,200.0.0.0/5,
208.0.0.0/4,224.0.0.0/3,::0/0
```

The same, but don't route IPv6 through the tunnel (note the last network in the output):

```
$ ./generate-wireguard-allowed-qr.py -4 --exclude 192.168.1.0/24 192.168.2.0/24
AllowedIPs = 0.0.0.0/1,128.0.0.0/2,192.0.0.0/9,192.128.0.0/11,192.160.0.0/13,192.168.0.0/24,192.168.3.0/24,
192.168.4.0/22,192.168.8.0/21,192.168.16.0/20,192.168.32.0/19,192.168.64.0/18,192.168.128.0/17,192.169.0.0/16,
192.170.0.0/15,192.172.0.0/14,192.176.0.0/12,192.192.0.0/10,193.0.0.0/8,194.0.0.0/7,196.0.0.0/6,200.0.0.0/5,
208.0.0.0/4,224.0.0.0/3
```

Allow only 192.168.1.1 and 192.168.1.2 to bypass the tunnel:

```
$ ./generate-wireguard-allowed-qr.py --exclude 192.168.1.1 192.168.1.2
AllowedIPs = 0.0.0.0/1,128.0.0.0/2,192.0.0.0/9,192.128.0.0/11,192.160.0.0/13,192.168.0.0/24,192.168.1.0/32,
192.168.1.3/32,192.168.1.4/30,192.168.1.8/29,192.168.1.16/28,192.168.1.32/27,192.168.1.64/26,192.168.1.128/25,
192.168.2.0/23,192.168.4.0/22,192.168.8.0/21,192.168.16.0/20,192.168.32.0/19,192.168.64.0/18,192.168.128.0/17,
192.169.0.0/16,192.170.0.0/15,192.172.0.0/14,192.176.0.0/12,192.192.0.0/10,193.0.0.0/8,194.0.0.0/7,196.0.0.0/6,
200.0.0.0/5,208.0.0.0/4,224.0.0.0/3,::0/0
```

The same, but also generate a QR code on the terminal that can be scanned by a mobile device. Note that when using `--qr`, the `AllowedIPs =` literal will not be printed as it is not required on a mobile device

```
$ ./generate-wireguard-allowed-qr.py --no-csv-output --qr --exclude 192.168.1.1 192.168.1.2
...
<QR CODE>
...
```

NOTE: When using QR code, the 'AllowedIPs =' string literal will not be included in the QR data by design, so that the data can be pasted into the value box on the mobile app **IMPORTANT**: You may also need to specify the WireGuard® peer address as an exclusion, you should test this

QR Code Support Dependencies

There are two ways to generate QR codes. By default, the script will try to use the `qrcode` Python module if available. If not available, it will fall back to using the `qrencode` application if it is present on your system. If neither are found, it will report an error

(RECOMMENDED) Python Package: qrcode

The qrcode is a simple, excellent QR code generation Python package that supports output to terminal windows as well as image files

(RECOMMENDED) Install qrcode via pip

Use `pip install 'qrcode[]'` or `pip install -r requirements.txt`. This will allow QR code generation without any native applications, such as the `qrencode` application

Install qrcode With Your OS Package Manager

This is not recommended as many distributions have dated versions of Python modules in their repositories. The distribution package is probably called `python3-qrcode`

OS Native / Package: qrencode

Use your distribution package manager to install `qrencode`

Debian / Ubuntu / Kali

```
$ sudo apt-get install qrencode
```

RHEL, Fedora, CentOS

```
$ sudo yum install qrencode
```

ArchLinux

```
$ sudo pacman -S qrencode
```

Other Options - Choose Your Own Adventure (TM)

If you would like to use a different QR code generator, you can change the `QR_ENCODE_COMMAND` list in the script, or you can add code to use some other Python library. This isn't recommended

TODO

- Perform additional testing

License / Copyright

```
(C) 2020, copyright@mzpqnxow.com BSD 3-Clause License
For terms, please see COPYING file
```

WireGuard® License

```
WireGuard® Copyright 2015-2020 Jason A. Donenfeld
All Rights Reserved
"WireGuard" and the "WireGuard" logo are registered trademarks of Jason A. Donenfeld.
```