TOR SUPPORT IN BITCOIN

It is possible to run Bitcoin Core as a Tor onion service, and connect to such services.

The following directions assume you have a Tor proxy running on port 9050. Many distributions default to having a SOCKS proxy listening on port 9050, but others may not. In particular, the Tor Browser Bundle defaults to listening on port 9150. See Tor Project FAQ:TBBSocksPort for how to properly configure Tor.

Compatibility

- Starting with version 22.0, Bitcoin Core only supports Tor version 3 hidden services (Tor v3). Tor v2 addresses are ignored by Bitcoin Core and neither relayed nor stored.
- Tor removed v2 support beginning with version 0.4.6.

How to see information about your Tor configuration via Bitcoin Core

There are several ways to see your local onion address in Bitcoin Core:

- in the "Local addresses" output of CLI `-netinfo`
- in the "localaddresses" output of RPC `getnetworkinfo`
- in the debug log (grep for "AddLocal"; the Tor address ends in `.onion`)

You may set the `-debug=tor` config logging option to have additional information in the debug log about your Tor configuration.

CLI `-addrinfo` returns the number of addresses known to your node per network. This can be useful to see how many onion peers your node knows, e.g. for `-onlynet=onion`.

To fetch a number of onion addresses that your node knows, for example seven addresses, use the `getnodeaddresses 7 onion` RPC.

1. Run Bitcoin Core behind a Tor proxy

The first step is running Bitcoin Core behind a Tor proxy. This will already anonymize all outgoing connections, but more is possible.

```
-proxy=ip:port  Set the proxy server. If SOCKS5 is selected (default), this proxy
                server will be used to try to reach .onion addresses as well.
                You need to use -noonion or -onion=0 to explicitly disable
                outbound access to onion services.

-onion=ip:port  Set the proxy server to use for Tor onion services. You do not
                need to set this if it's the same as -proxy. You can use -onion=0
                to explicitly disable access to onion services.
                ------------------------------------------------------------------
                Note: Only the -proxy option sets the proxy for DNS requests;
                with -onion they will not route over Tor, so use -proxy if you
                have privacy concerns.
                ------------------------------------------------------------------

-listen         When using -proxy, listening is disabled by default. If you want
```

```
                to manually configure an onion service (see section 3), you'll
                need to enable it explicitly.

-connect=X      When behind a Tor proxy, you can specify .onion addresses instead
-addnode=X      of IP addresses or hostnames in these parameters. It requires
-seednode=X     SOCKS5. In Tor mode, such addresses can also be exchanged with
                other P2P nodes.

-onlynet=onion  Make automatic outbound connections only to .onion addresses.
                Inbound and manual connections are not affected by this option.
                It can be specified multiple times to allow multiple networks,
                e.g. onlynet=onion, onlynet=i2p, onlynet=cjdns.
```

In a typical situation, this suffices to run behind a Tor proxy:

```
./bitcoind -proxy=127.0.0.1:9050
```

2. Automatically create a Bitcoin Core onion service

Bitcoin Core makes use of Tor's control socket API to create and destroy ephemeral onion services programmatically. This means that if Tor is running and proper authentication has been configured, Bitcoin Core automatically creates an onion service to listen on. The goal is to increase the number of available onion nodes.

This feature is enabled by default if Bitcoin Core is listening (`-listen`) and it requires a Tor connection to work. It can be explicitly disabled with `-listenonion=0` . If it is not disabled, it can be configured using the `-torcontrol` and `-torpassword` settings.

To see verbose Tor information in the bitcoind debug log, pass `-debug=tor` .

Control Port

You may need to set up the Tor Control Port. On Linux distributions there may be some or all of the following settings in `/etc/tor/torrc` , generally commented out by default (if not, add them):

```
ControlPort 9051
CookieAuthentication 1
CookieAuthFileGroupReadable 1
```

Add or uncomment those, save, and restart Tor (usually `systemctl restart tor` or `sudo systemctl restart tor` on most systemd-based systems, including recent Debian and Ubuntu, or just restart the computer).

On some systems (such as Arch Linux), you may also need to add the following line:

```
DataDirectoryGroupReadable 1
```

Authentication

Connecting to Tor's control socket API requires one of two authentication methods to be configured: cookie authentication or bitcoind's `-torpassword` configuration option.

Cookie authentication

For cookie authentication, the user running bitcoind must have read access to the `CookieAuthFile` specified in the Tor configuration. In some cases this is preconfigured and the creation of an onion service is automatic. Don't forget to use the `-debug=tor` bitcoind configuration option to enable Tor debug logging.

If a permissions problem is seen in the debug log, e.g. `tor: Authentication cookie /run/tor/control.authcookie could not be opened (check permissions)`, it can be resolved by adding both the user running Tor and the user running bitcoind to the same Tor group and setting permissions appropriately.

On Debian-derived systems, the Tor group will likely be `debian-tor` and one way to verify could be to list the groups and grep for a "tor" group name:

```
getent group | cut -d: -f1 | grep -i tor
```

You can also check the group of the cookie file. On most Linux systems, the Tor auth cookie will usually be `/run/tor/control.authcookie`:

```
TORGROUP=$(stat -c '%G' /run/tor/control.authcookie)
```

Once you have determined the `${TORGROUP}` and selected the `${USER}` that will run bitcoind, run this as root:

```
usermod -a -G ${TORGROUP} ${USER}
```

Then restart the computer (or log out) and log in as the `${USER}` that will run bitcoind.

`torpassword` authentication

For the `-torpassword=password` option, the password is the clear text form that was used when generating the hashed password for the `HashedControlPassword` option in the Tor configuration file.

The hashed password can be obtained with the command `tor --hash-password password` (refer to the Tor Dev Manual for more details).

3. Manually create a Bitcoin Core onion service

You can also manually configure your node to be reachable from the Tor network. Add these lines to your `/etc/tor/torrc` (or equivalent config file):

```
HiddenServiceDir /var/lib/tor/bitcoin-service/
HiddenServicePort 8333 127.0.0.1:8334
```

The directory can be different of course, but virtual port numbers should be equal to your bitcoind's P2P listen port (8333 by default), and target addresses and ports should be equal to binding address and port for inbound Tor connections (127.0.0.1:8334 by default).

```
-externalip=X   You can tell bitcoin about its publicly reachable addresses using
                this option, and this can be an onion address. Given the above
                configuration, you can find your onion address in
                /var/lib/tor/bitcoin-service/hostname. For connections
                coming from unroutable addresses (such as 127.0.0.1, where the
                Tor proxy typically runs), onion addresses are given
                preference for your node to advertise itself with.
```

```
                You can set multiple local addresses with -externalip. The
                one that will be rumoured to a particular peer is the most
                compatible one and also using heuristics, e.g. the address
                with the most incoming connections, etc.

-listen         You'll need to enable listening for incoming connections, as this
                is off by default behind a proxy.

-discover       When -externalip is specified, no attempt is made to discover local
                IPv4 or IPv6 addresses. If you want to run a dual stack, reachable
                from both Tor and IPv4 (or IPv6), you'll need to either pass your
                other addresses using -externalip, or explicitly enable -discover.
                Note that both addresses of a dual-stack system may be easily
                linkable using traffic analysis.
```

In a typical situation, where you're only reachable via Tor, this should suffice:

```
./bitcoind -proxy=127.0.0.1:9050 -
externalip=7zvj7a2imdgkdbg4f2dryd5rgtrn7upivr5eeij4cicjh65pooxeshid.onion -listen
```

(obviously, replace the .onion address with your own). It should be noted that you still listen on all devices and another node could establish a clearnet connection, when knowing your address. To mitigate this, additionally bind the address of your Tor proxy:

```
./bitcoind ... -bind=127.0.0.1
```

If you don't care too much about hiding your node, and want to be reachable on IPv4 as well, use `discover` instead:

```
./bitcoind ... -discover
```

and open port 8333 on your firewall (or use port mapping, i.e., `-upnp` or `-natpmp`).

If you only want to use Tor to reach .onion addresses, but not use it as a proxy for normal IPv4/IPv6 communication, use:

```
./bitcoind -onion=127.0.0.1:9050 -
externalip=7zvj7a2imdgkdbg4f2dryd5rgtrn7upivr5eeij4cicjh65pooxeshid.onion -discover
```

4. Privacy recommendations

- Do not add anything but Bitcoin Core ports to the onion service created in section 3. If you run a web service too, create a new onion service for that. Otherwise it is trivial to link them, which may reduce privacy. Onion services created automatically (as in section 2) always have only one port open.