Tunnel

Secure, multiplexed, TCP/UDP port forwarder using piping-server by @nwtgck as relay. Designed mainly for p2p connections between peers behind (multiple) NAT/firewalls.

Features

1. TCP/UDP tunnel between peers, each of which may be behind (multiple) NAT(s), i.e. unreachable from the public internet.
2. Firewalls don't cause problems as only outgoing http(s) connections are used.
3. Security: To connect, peers must know the unique ID of the serving peer and a shared secret key. Traffic between peer and relay is encrypted (TLS). Relay doesn't store anything.
4. Multiplexing: Each tunnel supports multiple concurrent connections. Connections are full-duplex.
5. Many-to-One: The forwarding peer acts as the client and the forwardee peer acts as the server. Server can support multiple clients at any given time. Each node can act as both server and client.
6. Resilience: Peers auto-reconnect in the face of intermittent connectivity.
7. No superuser privilege required.
8. Option to host your own relay server (easily and for free).
9. KISS: Just a single, small, portable, shell-script.
10. Built in installer and updater.

Command-line

ID: Every node is given a unique (base64) ID -

```
tunnel -i
```

ID is bound to hardware (MAC address) and the environment variables USER, HOME and HOSTNAME. Share it with your peers once and for all. Note: two users on the same machine are given separate node-IDs because their USER and HOME variables differ.

Server mode: Expose your local port to peers with whom you share any secret string -

```
tunnel [options] [-u] [-k <shared-secret>] <local-port>
```

Client mode: Forward your local port to peer's exposed local port -

```
tunnel [options] [-u] [-k <shared-secret>] [-b <local-port>] <peer-ID:peer-port>
```

If no local-port is provided using the `-b` option, `tunnel` uses a random unused port. The port used, is always reported at stdout.

Client and server must use the same secret to be able to connect with each other. The secret string may also be passed using the environment variable `TUNNEL_KEY`. Secret passed with `[-k]` takes precedence.

`[-u]` flag denotes use of UDP instead of the default TCP. If used, it must be used by both the peers.

All logs are at stderr by default.

Options:

-v Version

-h Help

-p *"<piping-server URL>"*

Can use environment variable `TUNNEL_RELAY` instead. Default: https://ppng.io. See list.

-l *"<path to log file>"*

Runs `tunnel` as daemon with stderr redirected to given path.

Installation and Updating

Download with:

```
curl -LO https://raw.githubusercontent.com/SomajitDey/tunnel/main/tunnel
```

Make it executable:

```
chmod +x ./tunnel
```

Then install system-wide with:

```
./tunnel -c install
```

If you don't have `sudo` privilege, you can install locally instead:

```
./tunnel -c install -l
```

To update anytime after installation:

```
tunnel -c update
```

Dependency/Portability

This program is simply an executable `bash` script depending on standard GNU tools including `socat`, `openssl`, `curl`, `mktemp`, `cut`, `awk`, `sed`, `flock`, `pkill`, `dd`, `xxd`, `base64` etc. that are readily available on standard Linux distros.

If your system lacks any of these tools, and you do not have the `sudo` privilege required to install it from the native package repository (e.g. `sudo apt-get install <package>`), try downloading a portable binary and install it locally at `${HOME}/.bin`.

Examples

SSH:

Peer A exposes local SSH port -

```
tunnel -k "${secret}" 22
```

Peer B connects -

```
tunnel -b 67868 -k "${secret}" -l /dev/null "${peerA_ID}:22" # Daemon due to -l
ssh -l "${login_name}" -p 67868 localhost
```

IPFS:

Let peer A has IPFS-peer-ID: `12orQmAlphanumeric`. Her IPFS daemon listens at default TCP port 4001. She exposes it with -

```
tunnel -k "${swarm_key}" 4001
```

Peer B now connects with peer A for file-sharing or pubsub or p2p -

```
port=$(TUNNEL_KEY="${swarm_key}" tunnel -l /dev/null "${peerA_ID}:4001")
ipfs swarm connect /ip4/127.0.0.1/tcp/${port}/p2p/12orQmAlphanumeric
```

Remote Shell:

Suppose you would regularly need to launch commands at your workplace Linux box from your home machine. And you don't want to / can't use SSH over `tunnel` for some reason.

At the workplace computer, expose some random local TCP port, e.g. 49090 and connect a shell to that port:

```
tunnel -l "/tmp/tunnel.log" -k "your secret" 49090 # Note the base64 node id
emitted
socat TCP-LISTEN:49090,reuseaddr,fork SYSTEM:'bash 2>&1'
```

Back at your home:

```
tunnel -l "/dev/null" -b 5000 -k "your secret" "node_id_of_workplace:49090"
rlwrap nc localhost 5000
```

Using rlwrap is not a necessity. But it sure makes the experience sweeter as it uses GNU Readline and remembers the input history (accessible with the up/down arrow keys similar to your local bash sessions).

Redis:

Need to connect to a remote Redis instance hosted by a peer or yourself? At the remote host, expose the TCP port that `redis-server` runs on (default: 6379), with `tunnel`.

At your local machine, use `tunnel` to forward a TCP port to the remote port. Point your `redis-cli` at the forwarded local port.

Applications

Below are some random use-cases I could think of for `tunnel`. Broadly speaking, anything that involves NAT/firewall traversal (e.g. WebRTC without TURN) or joining a remote LAN, should find `tunnel` useful. Some of the following ideas are rather sketchy, haven't been tested at all, and may not work, but nonetheless are documented here, at least for the time being, just for the sake of inspiration. If you found any of these useful, or useless, or you have found entirely new applications for `tunnel`, please post at discussions. Those cases that I have tested are labelled as "working".

- Connecting to IPFS peers (*Working*).
- P2P chatting, mailing, VoIP, streaming, gaming, screen-sharing, file-sharing, gambling, troubleshooting and what not.
- Connecting IOT devices.
- Connecting your workstation with your home-computer or laptop with SSH (*Working*), RDP or VNC.
- Shell-shovelling (*Working*).
- Beat your firewall with a self-hosted or peer-provided VPN.
- Joining intranet office chats (over office LAN) from your home across the internet. For example, BeeBEEP and LAN Messenger may use a local port that has been forwarded to from a node inside your office using `tunnel`.
- P2P (serverless) audio/video. For example, forward a local TCP/UDP port to peer's localhost and point your p2p client to it. Ready-made FOSS: Jami, Jitsi, Toxchat and Retroshare.
- Serverless remote-control using Teamviewer or Anydesk. Forward a local port, say 6000, to peer's local TCP port 5938 (for TV) and 7070 (for Anydesk) and use *127.0.0.1:6000* as the *IP:port* tuple of the peer.
- Making a local (web) port publicly accessible, *a.k.a* reverse-proxy accessible from the internet: Simply run `tunnel` at Heroku (for free) and forward the port stored in the environment variable `PORT` to your local port that you want to expose. And you have your public URL as: https://your-app-name.herokuapp.com.
- Accessing feed from a remote IP webcam using a universal network camera adapter: Forward a local port to a remote node that can access the mobile cam over WLAN.
- Streaming with VLC: Movies and Music, Webcam [command-line]. Just connect the server and client using `tunnel`.

Security

`tunnel` encrypts all traffic between a peer and the relay with TLS, if the relay uses https. There is no end-to-end encryption *per se* between the peers themselves. However, the piping-server relay is claimed to be *storageless*.

A client peer can connect with a serving peer only if they use the same secret key (TUNNEL_KEY). The key is primarily used for peer discovery at the relay stage. For every new connection to the forwarded local port, the client sends a random session key to the serving peer. The peers then form a new connection at another relay point based on this random key for the actual data transfer to occur. Outsiders, viz. bad actors who don't know the TUNNEL_KEY shouldn't be able to disrupt this flow.

However, a malicious peer can do the following. Because he knows the TUNNEL_KEY and the node ID of the serving peer, he can impersonate the latter. Data from an unsuspecting connecting peer, therefore, would be forwarded to the impersonator, starving the genuine server. Future updates/implementations of `tunnel` should handle this threat using public key crypto. [In that case, the random session key generated for every new connection to be forwarded, would be decryptable by the genuine server alone].

Given that `tunnel` is essentially the transport layer, the above points should not be discouraging, because most applications such as SSH and IPFS encrypt data at the application layer. Encrypting `tunnel` end-to-end for *all* data transfers would only add to the latency. However, you can always create an SSH-tunnel after establishing the low-level peering with `tunnel`, if you so choose.

Relay

The default relay used by `tunnel` is https://ppng.io. You can also use some other public relay from this list or host your own instance on free services such as offered by Heroku. Needless to say, to connect, two peers must use the same relay.

If you so choose, you can also write your own relay to be used by `tunnel` using simple tools like sertain. Just make sure your relay service has the same API as piping-server. If your relay code is open source, you are most welcome to introduce it at discussions.

See also

gsocket ; ipfs p2p with circuit-relay enabled ; go-piping-duplex ; pipeto.me ; uplink ; localhost.run ; ngrok ; localtunnel ; sshreach.me (*free trial for limited period only*) ; more

Notes:

1. Unlike piping-server, most of these do not offer easy, free self-hosting of the all-important relay or reverse proxy. If these services ever go down, you are doomed. With `tunnel` and piping-server, however, you can simply deploy your own relay instance, share its public URL with your peers once and for all, `export` the same as `TUNNEL_RELAY` inside `.bashrc` and you are good to go. Also, multiple public piping-servers are available for redundancy.
2. Some of these services, in the free tier, give a new random public URL for every session, which is problematic for intermittent connectivity (in IOT applications for example). Some free plans also expire sessions after a certain time even if connections are not idling.
3. Some expose your local port for web-traffic only. The onus of transporting non-web protocols over HTTP is left on you and your peers.

Future directions

IPFS:

Connecting to IPFS would be much simpler:

`tunnel -k <secret> ipfs` to expose and `tunnel -k <secret> <IPFS_peerID>` to connect.

These will launch the IPFS daemon on their own, if offline. The latter command will repeatedly swarm connect to the given peer at 30s intervals. The IPFS-peer-ID will be used as the node ID, so peers would no more need to share their node IDs separately. Non-default IPFS repo paths may be passed with option `-r`. or `IPFS_PATH`.

SSH:

Creating an SSH tunnel between local and peer port would be as easy as:

`tunnel -k <secret> ssh` to expose &

`tunnel -sk <secret> -b <local-port> <peerID>:<peer-port>` to create.

Note that, while connecting, one no more needs to provide a login name. The `${USER}` of the serving node is taken as the login name by default. However, if needed, a non-default login name can always be passed using an environment variable or option.

GPG:

Virtual machines, such as used by cloud-shells and dynos, do not have persistent, unique hardware addresses. The node ID therefore keeps on changing from session to session for such a VM. Future `tunnel` would have a `-g` option which would pass a GPG private key to `tunnel`. The node ID would be generated from the fingerprint of this key, akin to what IPFS does. This would also make `tunnel` more secure.

Argon2:

Option [`-a`] to use argon2 for hashing TUNNEL_KEY before use, so that a weaker secret isn't too vulnerable.

Bug-reports and Feedbacks

Please report bugs at issues. Post your thoughts, comments, ideas, use-cases and feature-requests at discussion. Let me know how this helped you, if it did at all.

Also feel free to write to me directly about anything regarding this project.

If this little script is of any use to you, a star would be immensely encouraging for me.

Thanks ! 😃
