sshuttle documentation

Release 1.0.5

Brian May

Dec 28, 2020

Contents

Date Dec 28, 2020

Version 1.0

Contents:

CHAPTER 1

Overview

As far as I know, sshuttle is the only program that solves the following common case:

- Your client machine (or router) is Linux, MacOS, FreeBSD, OpenBSD or pfSense.
- You have access to a remote network via ssh.
- You don't necessarily have admin access on the remote network.
- The remote network has no VPN, or only stupid/complex VPN protocols (IPsec, PPTP, etc). Or maybe you *are* the admin and you just got frustrated with the awful state of VPN tools.
- You don't want to create an ssh port forward for every single host/port on the remote network.
- You hate openssh's port forwarding because it's randomly slow and/or stupid.
- You can't use openssh's PermitTunnel feature because it's disabled by default on openssh servers; plus it does TCP-over-TCP, which has terrible performance (see below).

CHAPTER 2

Requirements

2.1 Client side Requirements

- sudo, or root access on your client machine. (The server doesn't need admin access.)
- Python 3.6 or greater.

2.1.1 Linux with NAT method

Supports:

- IPv4 TCP
- IPv4 DNS

Requires:

- iptables DNAT, REDIRECT, and ttl modules.

2.1.2 Linux with nft method

Supports

- IPv4 TCP
- IPv4 DNS
- IPv6 TCP
- IPv6 DNS

Requires:

- nftables

2.1.3 Linux with TPROXY method

Supports:

- IPv4 TCP
- IPv4 UDP (requires `recvmsg` - see below)
- IPv6 DNS (requires `recvmsg` - see below)
- IPv6 TCP
- IPv6 UDP (requires `recvmsg` - see below)
- IPv6 DNS (requires `recvmsg` - see below)

2.1.4 MacOS / FreeBSD / OpenBSD / pfSense

Method: pf

Supports:

- IPv4 TCP
- IPv4 DNS
- IPv6 TCP
- IPv6 DNS

Requires:

- You need to have the pfctl command.

2.1.5 Windows

Not officially supported, however can be made to work with Vagrant. Requires cmd.exe with Administrator access. See *Microsoft Windows* for more information.

2.2 Server side Requirements

- Python 3.6 or greater.

2.3 Additional Suggested Software

- If you are using systemd, sshuttle can notify it when the connection to the remote end is established and the firewall rules are installed. For this feature to work you must configure the process start-up type for the sshuttle service unit to notify, as shown in the example below.

```
[Unit]
Description=sshuttle
After=network.target

[Service]
Type=notify
```

(continues on next page)

(continued from previous page)

```
ExecStart=/usr/bin/sshuttle --dns --remote <user>@<server> <subnets...>

[Install]
WantedBy=multi-user.target
```

CHAPTER 3

Installation

- From PyPI:

```
pip install sshuttle
```

- Debain package manager:

```
sudo apt install sshuttle
```

- Clone:

```
git clone https://github.com/sshuttle/sshuttle.git
cd sshuttle
./setup.py install
```

3.1 Optionally after installation

- Add to sudoers file:

```
sshuttle --sudoers
```

CHAPTER 4

Usage

Note: For information on usage with Windows, see the *Microsoft Windows* section. For information on using the TProxy method, see the *TPROXY* section.

Forward all traffic:

```
sshuttle -r username@sshserver 0.0.0.0/0
```

- Use the *sshuttle -r* parameter to specify a remote server.
- By default sshuttle will automatically choose a method to use. Override with the *sshuttle --method* parameter.
- There is a shortcut for 0.0.0.0/0 for those that value their wrists:

  ```
  sshuttle -r username@sshserver 0/0
  ```

- For 'My VPN broke and need a temporary solution FAST to access local IPv4 addresses':

  ```
  sshuttle --dns -NHr username@sshserver 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16
  ```

If you would also like your DNS queries to be proxied through the DNS server of the server you are connect to:

```
sshuttle --dns -r username@sshserver 0/0
```

The above is probably what you want to use to prevent local network attacks such as Firesheep and friends. See the documentation for the *sshuttle --dns* parameter.

(You may be prompted for one or more passwords; first, the local password to become root using sudo, and then the remote ssh password. Or you might have sudo and ssh set up to not require passwords, in which case you won't be prompted at all.)

4.1 Usage Notes

That's it! Now your local machine can access the remote network as if you were right there. And if your "client" machine is a router, everyone on your local network can make connections to your remote network.

You don't need to install sshuttle on the remote server; the remote server just needs to have python available. sshuttle will automatically upload and run its source code to the remote python interpreter.

This creates a transparent proxy server on your local machine for all IP addresses that match 0.0.0.0/0. (You can use more specific IP addresses if you want; use any number of IP addresses or subnets to change which addresses get proxied. Using 0.0.0.0/0 proxies *everything*, which is interesting if you don't trust the people on your local network.)

Any TCP session you initiate to one of the proxied IP addresses will be captured by sshuttle and sent over an ssh session to the remote copy of sshuttle, which will then regenerate the connection on that end, and funnel the data back and forth through ssh.

Fun, right? A poor man's instant VPN, and you don't even have to have admin access on the server.

4.2 Sudoers File

sshuttle can auto-generate the proper sudoers.d file using the current user for Linux and OSX. Doing this will allow sshuttle to run without asking for the local sudo password and to give users who do not have sudo access ability to run sshuttle:

```
sshuttle --sudoers
```

DO NOT run this command with sudo, it will ask for your sudo password when it is needed.

A costume user or group can be set with the : option:*sshuttle –sudoers –sudoers-username {user_descriptor}* option. Valid values for this vary based on how your system is configured. Values such as usernames, groups pre-pended with % and sudoers user aliases will work. See the sudoers manual for more information on valid user specif actions. The options must be used with *–sudoers*:

```
sshuttle --sudoers --sudoers-user mike
sshuttle --sudoers --sudoers-user %sudo
```

The name of the file to be added to sudoers.d can be configured as well. This is mostly not necessary but can be useful for giving more than one user access to sshuttle. The default is *sshuttle_auto*:

```
sshuttle --sudoer --sudoers-filename sshuttle_auto_mike
sshuttle --sudoer --sudoers-filename sshuttle_auto_tommy
```

You can also see what configuration will be added to your system without modifying anything. This can be helpfull is the auto feature does not work, or you want more control. This option also works with *–sudoers-username*. *–sudoers-filename* has no effect with this option:

```
sshuttle --sudoers-no-modify
```

This will simply sprint the generated configuration to STDOUT. Example:

```
08:40 PM william$ sshuttle --sudoers-no-modify

Cmnd_Alias SSHUTTLE304 = /usr/bin/env PYTHONPATH=/usr/local/lib/python2.7/dist-
↪packages/sshuttle-0.78.5.dev30+gba5e6b5.d20180909-py2.7.egg /usr/bin/python /usr/
↪local/bin/sshuttle --method auto --firewall
```

(continues on next page)

(continued from previous page)

```
william ALL=NOPASSWD: SSHUTTLE304
```

CHAPTER 5

Platform Specific Notes

Contents:

5.1 Google ChromeOS

Currently there is no built in support for running sshuttle directly on Google ChromeOS/Chromebooks.

What we can really do is to create a Linux VM with Crostini. In the default stretch/Debian 9 VM, you can then install sshuttle as on any Linux box and it just works, as do xterms and ssvncviewer etc.

https://www.reddit.com/r/Crostini/wiki/getstarted/crostini-setup-guide

5.2 TPROXY

TPROXY is the only method that has full support of IPv6 and UDP.

There are some things you need to consider for TPROXY to work:

- The following commands need to be run first as root. This only needs to be done once after booting up:

```
ip route add local default dev lo table 100
ip rule add fwmark {TMARK} lookup 100
ip -6 route add local default dev lo table 100
ip -6 rule add fwmark {TMARK} lookup 100
```

  where {TMARK} is the identifier mark passed with -t or –tmark flag (default value is 1).

- The `--auto-nets` feature does not detect IPv6 routes automatically. Add IPv6 routes manually. e.g. by adding `'::/0'` to the end of the command line.

- The client needs to be run as root. e.g.:

```
sudo SSH_AUTH_SOCK="$SSH_AUTH_SOCK" $HOME/tree/sshuttle.tproxy/sshuttle  --
↪method=tproxy ...
```

(continues on next page)

(continued from previous page)

- You may need to exclude the IP address of the server you are connecting to. Otherwise sshuttle may attempt to intercept the ssh packets, which will not work. Use the `--exclude` parameter for this.
- Similarly, UDP return packets (including DNS) could get intercepted and bounced back. This is the case if you have a broad subnet such as `0.0.0.0/0` or `::/0` that includes the IP address of the client. Use the `--exclude` parameter for this.
- You need the `--method=tproxy` parameter, as above.
- The routes for the outgoing packets must already exist. For example, if your connection does not have IPv6 support, no IPv6 routes will exist, IPv6 packets will not be generated and sshuttle cannot intercept them:

```
telnet -6 www.google.com 80
Trying 2404:6800:4001:805::1010...
telnet: Unable to connect to remote host: Network is unreachable
```

  Add some dummy routes to external interfaces. Make sure they get removed however after sshuttle exits.

5.3 Microsoft Windows

Currently there is no built in support for running sshuttle directly on Microsoft Windows.

What we can really do is to create a Linux VM with Vagrant (or simply Virtualbox if you like). In the Vagrant settings, remember to turn on bridged NIC. Then, run sshuttle inside the VM like below:

```
sshuttle -l 0.0.0.0 -x 10.0.0.0/8 -x 192.168.0.0/16 0/0
```

10.0.0.0/8 excludes NAT traffic of Vagrant and 192.168.0.0/16 excludes traffic to local area network (assuming that we're using 192.168.0.0 subnet).

Assuming the VM has the IP 192.168.1.200 obtained on the bridge NIC (we can configure that in Vagrant), we can then ask Windows to route all its traffic via the VM by running the following in cmd.exe with admin right:

```
route add 0.0.0.0 mask 0.0.0.0 192.168.1.200
```

5.4 OpenWRT

Run:

```
opkg install python3 python3-pip iptables-mod-extra iptables-mod-nat-extra iptables-
↪mod-ipopt
python3 /usr/bin/pip3 install sshuttle
sshuttle -l 0.0.0.0 -r <IP> -x 192.168.1.1 0/0
```

CHAPTER 6

sshuttle

6.1 Synopsis

sshuttle [*options*] [**-r** *[username@]sshserver[:port]*] <*subnets ...*>

6.2 Description

sshuttle allows you to create a VPN connection from your machine to any remote server that you can connect to via ssh, as long as that server has python 3.6 or higher.

To work, you must have root access on the local machine, but you can have a normal account on the server.

It's valid to run **sshuttle** more than once simultaneously on a single client machine, connecting to a different server every time, so you can be on more than one VPN at once.

If run on a router, **sshuttle** can forward traffic for your entire subnet to the VPN.

6.3 Options

<subnets>
: A list of subnets to route over the VPN, in the form `a.b.c.d[/width][port[-port]]`. Valid examples are 1.2.3.4 (a single IP address), 1.2.3.4/32 (equivalent to 1.2.3.4), 1.2.3.0/24 (a 24-bit subnet, ie. with a 255.255.255.0 netmask), and 0/0 ('just route everything through the VPN'). Any of the previous examples are also valid if you append a port or a port range, so 1.2.3.4:8000 will only tunnel traffic that has as the destination port 8000 of 1.2.3.4 and 1.2.3.0/24:8000-9000 will tunnel traffic going to any port between 8000 and 9000 (inclusive) for all IPs in the 1.2.3.0/24 subnet. A hostname can be provided instead of an IP address. If the hostname resolves to multiple IPs, all of the IPs are included. If a width is provided with a hostname that the width is applied to all of the hostnames IPs (if they are all either IPv4 or IPv6). Widths cannot be supplied to hostnames that resolve to both IPv4 and IPv6. Valid examples are example.com, example.com:8000, example.com/24, example.com/24:8000 and example.com:8000-9000.

--method <auto|nat|nft|tproxy|pf|ipfw>
: Which firewall method should sshuttle use? For auto, sshuttle attempts to guess the appropriate method depending on what it can find in PATH. The default value is auto.

-l <[ip:]port>, **--listen**=<[ip:]port>
: Use this ip address and port number as the transparent proxy port. By default **sshuttle** finds an available port automatically and listens on IP 127.0.0.1 (localhost), so you don't need to override it, and connections are only proxied from the local machine, not from outside machines. If you want to accept connections from other machines on your network (ie. to run **sshuttle** on a router) try enabling IP Forwarding in your kernel, then using `--listen 0.0.0.0:0`. You can use any name resolving to an IP address of the machine running **sshuttle**, e.g. `--listen localhost`.

  For the nft, tproxy and pf methods this can be an IPv6 address. Use this option with comma separated values if required, to provide both IPv4 and IPv6 addresses, e.g. `--listen 127.0.0.1:0,[::1]:0`.

-H, --auto-hosts
: Scan for remote hostnames and update the local /etc/hosts file with matching entries for as long as the VPN is open. This is nicer than changing your system's DNS (/etc/resolv.conf) settings, for several reasons. First, hostnames are added without domain names attached, so you can `ssh thatserver` without worrying if your local domain matches the remote one. Second, if you **sshuttle** into more than one VPN at a time, it's impossible to use more than one DNS server at once anyway, but **sshuttle** correctly merges /etc/hosts entries between all running copies. Third, if you're only routing a few subnets over the VPN, you probably would prefer to keep using your local DNS server for everything else.

-N, --auto-nets
: In addition to the subnets provided on the command line, ask the server which subnets it thinks we should route, and route those automatically. The suggestions are taken automatically from the server's routing table.

  This feature does not detect IPv6 routes. Specify IPv6 subnets manually. For example, specify the `::/0` subnet on the command line to route all IPv6 traffic.

--dns
: Capture local DNS requests and forward to the remote DNS server. All queries to any of the local system's DNS servers (/etc/resolv.conf and, if it exists, /run/systemd/resolve/resolv.conf) will be intercepted and resolved on the remote side of the tunnel instead, there using the DNS specified via the *--to-ns* option, if specified. Only plain DNS traffic sent to these servers on port 53 are captured.

--ns-hosts=<server1[,server2[,server3[...]]]>
: Capture local DNS requests to the specified server(s) and forward to the remote DNS server. Contrary to the *--dns* option, this flag allows to specify the DNS server(s) the queries to which to intercept, instead of intercepting all DNS traffic on the local machine. This can be useful when only certain DNS requests should be resolved on the remote side of the tunnel, e.g. in combination with dnsmasq.

--to-ns=<server>
: The DNS to forward requests to when remote DNS resolution is enabled. If not given, sshuttle will simply resolve using the system configured resolver on the remote side (via /etc/resolv.conf on the remote side).

--python
: Specify the name/path of the remote python interpreter. The default is to use `python3` (or `python`, if `python3` fails) in the remote system's PATH.

-r <[username@]sshserver[:port]>, **--remote**=<[username@]sshserver[:port]>
: The remote hostname and optional username and ssh port number to use for connecting to the remote server. For example, example.com, testuser@example.com, testuser@example.com:2222, or example.com:2244.

-x <subnet>, **--exclude**=<subnet>
: Explicitly exclude this subnet from forwarding. The format of this option is the same as the <subnets> option. To exclude more than one subnet, specify the `-x` option more than once. You can say something like `0/0 -x 1.2.3.0/24` to forward everything except the local subnet over the VPN, for example.

-X <file>, **--exclude-from**=<file>
Exclude the subnets specified in a file, one subnet per line. Useful when you have lots of subnets to exclude.

-v, --verbose
Print more information about the session. This option can be used more than once for increased verbosity. By default, **sshuttle** prints only error messages.

-e, --ssh-cmd
The command to use to connect to the remote server. The default is just `ssh`. Use this if your ssh client is in a non-standard location or you want to provide extra options to the ssh command, for example, `-e 'ssh -v'`.

--seed-hosts
A comma-separated list of hostnames to use to initialize the *--auto-hosts* scan algorithm. *--auto-hosts* does things like poll local SMB servers for lists of local hostnames, but can speed things up if you use this option to give it a few names to start from.

If this option is used *without* *--auto-hosts*, then the listed hostnames will be scanned and added, but no further hostnames will be added.

--no-latency-control
Sacrifice latency to improve bandwidth benchmarks. ssh uses really big socket buffers, which can overload the connection if you start doing large file transfers, thus making all your other sessions inside the same tunnel go slowly. Normally, **sshuttle** tries to avoid this problem using a "fullness check" that allows only a certain amount of outstanding data to be buffered at a time. But on high-bandwidth links, this can leave a lot of your bandwidth underutilized. It also makes **sshuttle** seem slow in bandwidth benchmarks (benchmarks rarely test ping latency, which is what **sshuttle** is trying to control). This option disables the latency control feature, maximizing bandwidth usage. Use at your own risk.

--latency-buffer-size
Set the size of the buffer used in latency control. The default is `32768`. Changing this option allows a compromise to be made between latency and bandwidth without completely disabling latency control (with *--no-latency-control*).

-D, --daemon
Automatically fork into the background after connecting to the remote server. Implies *--syslog*.

-s <file>, **--subnets**=<file>
Include the subnets specified in a file instead of on the command line. One subnet per line.

--syslog
after connecting, send all log messages to the *syslog(3)* service instead of stderr. This is implicit if you use *--daemon*.

--pidfile=<pidfilename>
when using *--daemon*, save **sshuttle**'s pid to *pidfilename*. The default is `sshuttle.pid` in the current directory.

--disable-ipv6
Disable IPv6 support for methods that support it (nft, tproxy, and pf).

--firewall
(internal use only) run the firewall manager. This is the only part of **sshuttle** that must run as root. If you start **sshuttle** as a non-root user, it will automatically run `sudo` or `su` to start the firewall manager, but the core of **sshuttle** still runs as a normal user.

--hostwatch
(internal use only) run the hostwatch daemon. This process runs on the server side and collects hostnames for the *--auto-hosts* option. Using this option by itself makes it a lot easier to debug and test the *--auto-hosts* feature.

--sudoers
sshuttle will auto generate the proper sudoers.d config file and add it. Once this is completed, sshuttle will exit and tell the user if it succeed or not. Do not call this options with sudo, it may generate a incorrect config file.

--sudoers-no-modify
sshuttle will auto generate the proper sudoers.d config and print it to stdout. The option will not modify the system at all.

--sudoers-user
Set the user name or group with %group_name for passwordless operation. Default is the current user.set ALL for all users. Only works with –sudoers or –sudoers-no-modify option.

--sudoers-filename
Set the file name for the sudoers.d file to be added. Default is "sshuttle_auto". Only works with –sudoers.

-t, --tmark
Transproxy optional traffic mark with provided MARK value.

--version
Print program version.

6.4 Configuration File

All the options described above can optionally be specified in a configuration file.

To run **sshuttle** with options defined in, e.g., */etc/sshuttle.conf* just pass the path to the file preceded by the @ character, e.g. *@/etc/sshuttle.conf*.

When running **sshuttle** with options defined in a configuration file, options can still be passed via the command line in addition to what is defined in the file. If a given option is defined both in the file and in the command line, the value in the command line will take precedence.

Arguments read from a file must be one per line, as shown below:

```
value
--option1
value1
--option2
value2
```

6.5 Examples

Test locally by proxying all local connections, without using ssh:

```
$ sshuttle -v 0/0

Starting sshuttle proxy.
Listening on ('0.0.0.0', 12300).
[local sudo] Password:
firewall manager ready.
c : connecting to server...
 s: available routes:
 s:   192.168.42.0/24
c : connected.
firewall manager: starting transproxy.
```

(continues on next page)

(continued from previous page)

```
c : Accept: 192.168.42.106:50035 -> 192.168.42.121:139.
c : Accept: 192.168.42.121:47523 -> 77.141.99.22:443.
    ...etc...
^C
firewall manager: undoing changes.
KeyboardInterrupt
c : Keyboard interrupt: exiting.
c : SW#8:192.168.42.121:47523: deleting
c : SW#6:192.168.42.106:50035: deleting
```

Test connection to a remote server, with automatic hostname and subnet guessing:

```
$ sshuttle -vNHr example.org

Starting sshuttle proxy.
Listening on ('0.0.0.0', 12300).
firewall manager ready.
c : connecting to server...
 s: available routes:
 s:   77.141.99.0/24
c : connected.
c : seed_hosts: []
firewall manager: starting transproxy.
hostwatch: Found: testbox1: 1.2.3.4
hostwatch: Found: mytest2: 5.6.7.8
hostwatch: Found: domaincontroller: 99.1.2.3
c : Accept: 192.168.42.121:60554 -> 77.141.99.22:22.
^C
firewall manager: undoing changes.
c : Keyboard interrupt: exiting.
c : SW#6:192.168.42.121:60554: deleting
```

Run **sshuttle** with a */etc/sshuttle.conf* configuration file:

```
$ sshuttle @/etc/sshuttle.conf
```

Use the options defined in */etc/sshuttle.conf* but be more verbose:

```
$ sshuttle @/etc/sshuttle.conf -vvv
```

Override the remote server defined in */etc/sshuttle.conf*:

```
$ sshuttle @/etc/sshuttle.conf -r otheruser@test.example.com
```

Example configuration file:

```
192.168.0.0/16
--remote
user@example.com
```

6.6 Discussion

When it starts, **sshuttle** creates an ssh session to the server specified by the `-r` option. If `-r` is omitted, it will start both its client and server locally, which is sometimes useful for testing.

After connecting to the remote server, **sshuttle** uploads its (python) source code to the remote end and executes it there. Thus, you don't need to install **sshuttle** on the remote server, and there are never **sshuttle** version conflicts between client and server.

Unlike most VPNs, **sshuttle** forwards sessions, not packets. That is, it uses kernel transparent proxying (*iptables REDIRECT* rules on Linux) to capture outgoing TCP sessions, then creates entirely separate TCP sessions out to the original destination at the other end of the tunnel.

Packet-level forwarding (eg. using the tun/tap devices on Linux) seems elegant at first, but it results in several problems, notably the 'tcp over tcp' problem. The tcp protocol depends fundamentally on packets being dropped in order to implement its congestion control agorithm; if you pass tcp packets through a tcp-based tunnel (such as ssh), the inner tcp packets will never be dropped, and so the inner tcp stream's congestion control will be completely broken, and performance will be terrible. Thus, packet-based VPNs (such as IPsec and openvpn) cannot use tcp-based encrypted streams like ssh or ssl, and have to implement their own encryption from scratch, which is very complex and error prone.

sshuttle's simplicity comes from the fact that it can safely use the existing ssh encrypted tunnel without incurring a performance penalty. It does this by letting the client-side kernel manage the incoming tcp stream, and the server-side kernel manage the outgoing tcp stream; there is no need for congestion control to be shared between the two separate streams, so a tcp-based tunnel is fine.

See also:

ssh(1), *python(1)*

CHAPTER 7

How it works

sshuttle is not exactly a VPN, and not exactly port forwarding. It's kind of both, and kind of neither.

It's like a VPN, since it can forward every port on an entire network, not just ports you specify. Conveniently, it lets you use the "real" IP addresses of each host rather than faking port numbers on localhost.

On the other hand, the way it *works* is more like ssh port forwarding than a VPN. Normally, a VPN forwards your data one packet at a time, and doesn't care about individual connections; ie. it's "stateless" with respect to the traffic. sshuttle is the opposite of stateless; it tracks every single connection.

You could compare sshuttle to something like the old Slirp program, which was a userspace TCP/IP implementation that did something similar. But it operated on a packet-by-packet basis on the client side, reassembling the packets on the server side. That worked okay back in the "real live serial port" days, because serial ports had predictable latency and buffering.

But you can't safely just forward TCP packets over a TCP session (like ssh), because TCP's performance depends fundamentally on packet loss; it *must* experience packet loss in order to know when to slow down! At the same time, the outer TCP session (ssh, in this case) is a reliable transport, which means that what you forward through the tunnel *never* experiences packet loss. The ssh session itself experiences packet loss, of course, but TCP fixes it up and ssh (and thus you) never know the difference. But neither does your inner TCP session, and extremely screwy performance ensues.

sshuttle assembles the TCP stream locally, multiplexes it statefully over an ssh session, and disassembles it back into packets at the other end. So it never ends up doing TCP-over-TCP. It's just data-over-TCP, which is safe.

CHAPTER 8

Support

Mailing list:

- Subscribe by sending a message to <sshuttle+subscribe@googlegroups.com>
- List archives are at: http://groups.google.com/group/sshuttle

Issue tracker and pull requests at github:

- https://github.com/sshuttle/sshuttle

CHAPTER 9

Useless Trivia

This section written by the original author, Avery Pennarun <apenwarr@gmail.com>.

Back in 1998, I released the first version of Tunnel Vision, a semi-intelligent VPN client for Linux. Unfortunately, I made two big mistakes: I implemented the key exchange myself (oops), and I ended up doing TCP-over-TCP (double oops). The resulting program worked okay - and people used it for years - but the performance was always a bit funny. And nobody ever found any security flaws in my key exchange, either, but that doesn't mean anything. :)

The same year, dcoombs and I also released Fast Forward, a proxy server supporting transparent proxying. Among other things, we used it for automatically splitting traffic across more than one Internet connection (a tool we called "Double Vision").

I was still in university at the time. A couple years after that, one of my professors was working with some graduate students on the technology that would eventually become Slipstream Internet Acceleration. He asked me to do a contract for him to build an initial prototype of a transparent proxy server for mobile networks. The idea was similar to sshuttle: if you reassemble and then disassemble the TCP packets, you can reduce latency and improve performance vs. just forwarding the packets over a plain VPN or mobile network. (It's unlikely that any of my code has persisted in the Slipstream product today, but the concept is still pretty cool. I'm still horrified that people use plain TCP on complex mobile networks with crazily variable latency, for which it was never really intended.)

That project I did for Slipstream was what first gave me the idea to merge the concepts of Fast Forward, Double Vision, and Tunnel Vision into a single program that was the best of all worlds. And here we are, at last. You're welcome.

CHAPTER 10

Change log

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

10.1 1.0.5 - 2020-12-29

10.1.1 Added

- IPv6 support in nft method.
- Intercept DNS requests sent by systemd-resolved.
- Set default tmark.
- Fix python2 server compatibility.
- Python 3.9 support.

10.1.2 Fixed

- Change license text to LGPL-2.1
- Fix #494 sshuttle caught in infinite select() loop.
- Include sshuttle version in verbose output.
- Add psutil as dependency in setup.py
- When subnets and excludes are specified with hostnames, use all IPs.
- Update/document client's handling of IPv4 and IPv6.
- Update sdnotify.py documentation.
- Allow no remote to work.

- Make prefixes in verbose output more consistent.
- Make nat and nft rules consistent; improve rule ordering.
- Make server and client handle resolv.conf differently.
- Fix handling OSError in FirewallClient#__init__
- Refactor automatic method selection.

10.1.3 Removed

- Drop testing of Python 3.5

10.2 1.0.4 - 2020-08-24

10.2.1 Fixed

- Allow Mux() flush/fill to work with python < 3.5
- Fix parse_hostport to always return string for host.
- Require -r/–remote parameter.
- Add missing package in OpenWRT documentation.
- Fix doc about –listen option.
- README: add Ubuntu.
- Increase IP4 ttl to 63 hops instead of 42.
- Fix formatting in installation.rst

10.3 1.0.3 - 2020-07-12

10.3.1 Fixed

- Ask setuptools to require Python 3.5 and above.
- Add missing import.
- Fix formatting typos in usage docs

10.4 1.0.2 - 2020-06-18

10.4.1 Fixed

- Leave use of default port to ssh command.
- Remove unwanted references to Python 2.7 in docs.
- Replace usage of deprecated imp.
- Fix connection with @ sign in username.

10.5 1.0.1 - 2020-06-05

10.5.1 Fixed

- Errors in python long_documentation.

10.6 1.0.0 - 2020-06-05

10.6.1 Added

- Python 3.8 support.
- sshpass support.
- Auto sudoers file (#269).
- option for latency control buffer size.
- Docs: FreeBSD'.
- Docs: Nix'.
- Docs: openwrt'.
- Docs: install instructions for Fedora'.
- Docs: install instructions for Arch Linux'.
- Docs: 'My VPN broke and need a solution fast'.

10.6.2 Removed

- Python 2.6 support.
- Python 2.7 support.

10.6.3 Fixed

- Remove debug message for getpeername failure.
- Fix crash triggered by port scans closing socket.
- Added "Running as a service" to docs.
- Systemd integration.
- Trap UnicodeError to handle cases where hostnames returned by DNS are invalid.
- Formatting error in CHANGES.rst
- Various errors in documentation.
- Nftables based method.
- Make hostwatch locale-independent (#379).
- Add tproxy udp port mark filter that was missed in #144, fixes #367.
- Capturing of local DNS servers.

- Crashing on ECONNABORTED.
- Size of pf_rule, which grew in OpenBSD 6.4.
- Use prompt for sudo, not needed for doas.
- Arch linux installation instructions.
- tests for existing PR-312 (#337).
- Hyphen in hostname.
- Assembler import (#319).

10.7 0.78.5 - 2019-01-28

10.7.1 Added

- doas support as replacmeent for sudo on OpenBSD.
- Added ChromeOS section to documentation (#262)
- Add –no-sudo-pythonpath option

10.7.2 Fixed

- Fix forwarding to a single port.
- Various updates to documentation.
- Don't crash if we can't look up peername
- Fix missing string formatting argument
- Moved sshuttle/tests into tests.
- Updated bandit config.
- Replace path /dev/null by os.devnull.
- Added coverage report to tests.
- Fixes support for OpenBSD (6.1+) (#282).
- Close stdin, stdout, and stderr when using syslog or forking to daemon (#283).
- Changes pf exclusion rules precedence.
- Fix deadlock with iptables with large ruleset.
- docs: document –ns-hosts –to-ns and update –dns.
- Use subprocess.check_output instead of run.
- Fix potential deadlock condition in nft_get_handle.
- auto-nets: retrieve routes only if using auto-nets.

10.8 0.78.4 - 2018-04-02

10.8.1 Added

- Add homebrew instructions.
- Route traffic by linux user.
- Add nat-like method using nftables instead of iptables.

10.8.2 Changed

- Talk to custom DNS server on pod, instead of the ones in /etc/resolv.conf.
- Add new option for overriding destination DNS server.
- Changed subnet parsing. Previously 10/8 become 10.0.0.0/8. Now it gets parsed as 0.0.0.10/8.
- Make hostwatch find both fqdn and hostname.
- Use versions of python3 greater than 3.5 when available (e.g. 3.6).

10.8.3 Removed

- Remove Python 2.6 from automatic tests.

10.8.4 Fixed

- Fix case where there is no –dns.
- [pf] Avoid port forwarding from loopback address.
- Use getaddrinfo to obtain a correct sockaddr.
- Skip empty lines on incoming routes data.
- Just skip empty lines of routes data instead of stopping processing.
- [pf] Load pf kernel module when enabling pf.
- [pf] Test double restore (ipv4, ipv6) disables only once; test kldload.
- Fixes UDP and DNS proxies binding to the same socket address.
- Mock socket bind to avoid depending on local IPs being available in test box.
- Fix no value passed for argument auto_hosts in hw_main call.
- Fixed incorrect license information in setup.py.
- Preserve peer and port properly.
- Make –to-dns and –ns-host work well together.
- Remove test that fails under OSX.
- Specify pip requirements for tests.
- Use flake8 to find Python syntax errors or undefined names.
- Fix compatibility with the sudoers file.

- Stop using SO_REUSEADDR on sockets.
- Declare 'verbosity' as global variable to placate linters.
- Adds 'cd sshuttle' after 'git' to README and docs.
- Documentation for loading options from configuration file.
- Load options from a file.
- Fix firewall.py.
- Move sdnotify after setting up firewall rules.
- Fix tests on Macos.

10.9 0.78.3 - 2017-07-09

The "I should have done a git pull" first release.

10.9.1 Fixed

- Order first by port range and only then by swidth

10.10 0.78.2 - 2017-07-09

10.10.1 Added

- Adds support for tunneling specific port ranges (#144).
- Add support for iproute2.
- Allow remote hosts with colons in the username.
- Re-introduce ipfw support for sshuttle on FreeBSD with support for –DNS option as well.
- Add support for PfSense.
- Tests and documentation for systemd integration.
- Allow subnets to be given only by file (-s).

10.10.2 Fixed

- Work around non tabular headers in BSD netstat.
- Fix UDP and DNS support on Python 2.7 with tproxy method.
- Fixed tests after adding support for iproute2.
- Small refactoring of netstat/iproute parsing.
- Set started_by_sshuttle False after disabling pf.
- Fix punctuation and explain Type=notify.
- Move pytest-runner to tests_require.
- Fix warning: closed channel got=STOP_SENDING.

- Support sdnotify for better systemd integration.
- Fix #117 to allow for no subnets via file (-s).
- Fix argument splitting for multi-word arguments.
- requirements.rst: Fix mistakes.
- Fix typo, space not required here.
- Update installation instructions.
- Support using run from different directory.
- Ensure we update sshuttle/version.py in run.
- Don't print python version in run.
- Add CWD to PYTHONPATH in run.

10.11 0.78.1 - 2016-08-06

- Fix readthedocs versioning.
- Don't crash on ENETUNREACH.
- Various bug fixes.
- Improvements to BSD and OSX support.

10.12 0.78.0 - 2016-04-08

- Don't force IPv6 if IPv6 nameservers supplied. Fixes #74.
- Call /bin/sh as users shell may not be POSIX compliant. Fixes #77.
- Use argparse for command line processing. Fixes #75.
- Remove useless –server option.
- Support multiple -s (subnet) options. Fixes #86.
- Make server parts work with old versions of Python. Fixes #81.

10.13 0.77.2 - 2016-03-07

- Accidentally switched LGPL2 license with GPL2 license in 0.77.1 - now fixed.

10.14 0.77.1 - 2016-03-07

- Use semantic versioning. http://semver.org/
- Update GPL 2 license text.
- New release to fix PyPI.

10.15 0.77 - 2016-03-03

- Various bug fixes.
- Fix Documentation.
- Add fix for MacOS X issue.
- Add support for OpenBSD.

10.16 0.76 - 2016-01-17

- Add option to disable IPv6 support.
- Update documentation.
- Move documentation, including man page, to Sphinx.
- Use setuptools-scm for automatic versioning.

10.17 0.75 - 2016-01-12

- Revert change that broke sshuttle entry point.

10.18 0.74 - 2016-01-10

- Add CHANGES.rst file.
- Numerous bug fixes.
- Python 3.5 fixes.
- PF fixes, especially for BSD.

CHAPTER 11

Indices and tables

- genindex
- search

Index

Symbols

S