

Steffen Wendzel¹

vstt [Very Strange Tunneling Tool] documentation

¹steffenwendzel -at- gmx -dot- net, www.wendzel.de

Contents

1 Disclaimer	3
2 Introduction	3
3 How to use it?	3
4 Examples	3
4.1 Example 1 (without a TCP connection)	4
4.2 Example 2 (tunneling a SSH connection)	4
5 Protocols	6
5.1 none	6
5.2 POP3 (alpha quality)	6
5.3 ICMP	6
6 Gateways	6
7 Comments, Feedback	6

1 Disclaimer

This tool is for legal purposes only! Please also read the LICENSE file for license details.

2 Introduction

`vstt` is a tunneling tool (primary for TCP connections). It can send your data via different protocols. Please send your patches if you port it to new systems or if you fixed a bug.

Currently tested systems are:

- Linux 2.6.x and newer (i386 or amd64)
- OpenBSD 3.x to 4.0-current (i386 and amd64)
- SHOULD work too: FreeBSD, NetBSD and Solaris (Solaris needs a Makefile modification)

`vstt` can tunnel your data within the following protocols:

- NONE (a pseudo protocol) - 99% done
- ICMP - 95% done
- POP3 - 90% done
- DNS - 5% done (only stub)

3 How to use it?

`vstt` receives input from a source, transfers it over a tunnel to another system running `vstt`, and outputs the received input to a destination.

`vstt` accepts input either from a local FIFO or from a TCP stream socket that you can bind to a port. Similarly, `vstt` outputs data to FIFO on the receiver or to a TCP stream socket that you bind to a port.

If you use local FIFOs for input/output, `vstt` uses the following files:

```
binary name | input fifo           | output fifo
-----
vstt        | /tmp/.vstt_send2peer | /tmp/.vstt_recvfpeer
```

You can send data into the connection by writing data into the input fifo and you can read received data from the peer via reading from the output fifo.

Q: But I want to use sockets because my TCP app (Telnet or SSH for example) uses TCP and not FIFOs.

A: No problem: you have to use the `s2f` tool included in the code -- it binds a TCP socket to a FIFO!

4 Examples

Note: `vstt` normally produces one 'connection refused' error every second if the other peer is not already available. The error messages will disappear once the connection is established.

4.1 Example 1 (without a TCP connection)

Let us create a simple ICMP tunnel using `vstt` on two machines. We want to send a file through the tunnel and read it with the shipped tool 'reader'.

This setup requires different parameters to start `vstt`:

```
-p icmp      <- set the protocol to pop3
-r n        <- receive data on port n (ignored with ICMP)
-t m        <- send data to the peer at port m (ignored with ICMP)
-a x        <- the IP address of the peer
-m y        <- own IP address
```

Setup: We use two Linux machines with the following IPs and will transfer a simple text file.

```
Sender: 192.168.2.102
Receiver: 192.168.2.101
Protocol: ICMP
```

On the sender, we run the following command (-r and -t are ignored on both computers but must be added):

```
sudo ./vstt -p icmp -r 9999 -t 10000 -a 192.168.2.101 -m 192.168.2.102
```

(This means to use ICMP; the sender's address is *102, the peer's address is *101)

On the receiver, we run:

```
sudo ./vstt2 -p icmp -r 10001 -t 10002 -a 192.168.2.102 -m 192.168.2.101
```

... and in another terminal on the receiver, we start `reader` that reads the received data from the FIFO:

```
sudo ./reader /tmp/.vstt_recvfpeer
```

Now the tunnel setup is complete. The data will be transferred via ICMP from sender to receiver and it will be read out from `/tmp/.vstt_recvfpeer`.

Now, we just need to send the actual data that we want to tunnel from the sender to the receiver.

On the sender, we simply send the input from a system configuration file to the pipe:

```
\$ sudo -i
# cat /etc/resolv.conf > /tmp/.vstt_send2peer
```

If we now look at the output of the `reader` on the receiver, we will see the content of `/etc/resolv.conf` that was tunneled via ICMP.

4.2 Example 2 (tunneling a SSH connection)

Now we want to tunnel a SSH connection between two hosts over port 80 (e.g. because a firewall does not block HTTP but SSH). We use the protocol 'none' because it's fast, works very well. 'none' creates nothing but a plain TCP-based tunnel.

Note: You need root access to bind ports less than 1024 under Unix(-like) systems.

This works as follows: both systems start a `vstt`-tunnel they can communicate through. On the SSH server we connect our `vstt`'s FIFO with the SSH service on port 22 (what can be done by using the `s2f` tool – very simple).

On the client, machine we use the tool `s2f`, too (but in server mode). `s2f` communicates with the local `vstt` endpoint via its FIFO. Finally, we connect to the `s2f` port using our local SSH client. That's all.

Okay, Let's start.

Say that 'eygo' (192.168.2.20) is the machine with the SSH-Server and that 'hikoki' (192.168.2.21) is the server with the SSH client.

On the first terminal (xterm or a console terminal or whatever), we start `vsst`. We receive data on port 80 and send data to port 80 at the other `vsst`-endpoint.

```
eygo# ./vsst -p none -r 80 -t 80 -a 192.168.2.20
client: connecting to peer ...
server: waiting for connection...
none(or pop3 and so on)_client: connect(): Connection refused
none(or pop3 and so on)_client: connect(): Connection refused
none(or pop3 and so on)_client: connect(): Connection refused
none(or pop3 and so on)_client: connect(): Connection refused
none(or pop3 and so on)_client: connect(): Connection refused
...
...
```

On the second terminal we start `s2f`. It will listen on port 10003. We will connect to this port with the ssh client if the tunnel works.

```
eygo# ./s2f -s -p 10003
```

IMPORTANT NOTE: If you don't want to start `s2f` by hand, you can also let `vsst` do that by using `-c <port> [-s]` parameters! Instead of starting `vsst+s2f`, you could start only `vsst` in this example:

```
# vsst -p none -r 80 -t 80 -a 192.168.2.20 -c 10003 -s
```

Please note that the parameter '-s' means to run as a server and to use the port given with `-p` as the listen port instead as the port to connect to.

On eygo, we start `vsst` too:

```
eygo# ./vsst -p none -r 80 -t 80 -a 192.168.2.21
client: connecting to peer ...
server: waiting for connection...
wrapper_tcpserver: connection established => waiting for data...
==> con establ
client: waiting for data from fifo...
```

And we connect the `vsst`-FIFOs to the local SSH-Server running on Port 22 via `s2f`:

```
eygo# ./s2f -p 22
connected.
```

IMPORTANT NOTE: You could alternativeley only start `vsst` one time without calling `s2f`:

```
# ./vsst -p none -r 80 -t 80 -a 192.168.2.21 -c 22
```

And now, you can connect with SSH to the localhost port 10003 on the first machine (hikoki).

```
hikoki$ ssh user@127.0.0.1 -p 10003
```

That's it. Your tunnel should now work.

5 Protocols

5.1 none

The ‘none’ protocol is used for a blank tunnel. For example: You sit behind a firewall that only lets you use port 80 but you want to connect to your IRC-server at home. You can use the ‘none’ protocol to redirect a connection over port 80 and then bypass the firewall and enjoy your IRC session.

5.2 POP3 (alpha quality)

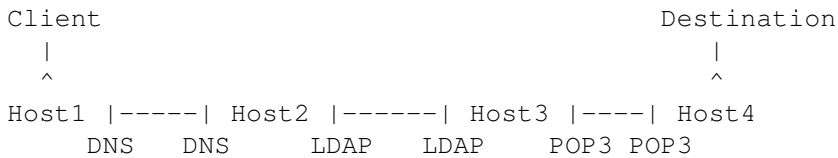
This is a little bit more advanced. A POP3 tunnel is slow but it can hide your data in RETR-requests. If you want to hide your data a little bit: use POP3 (or ICMP).

5.3 ICMP

If all TCP+UDP ports are blocked, an ICMP tunnel can work anyway. `vstt` sends your data as payload in ICMP echo datagrams. `vstt` can re-send lost packets, re-calculates the checksum to detect corrupted packets and can also send big packets from your applications within many small ICMP packets that will be re-assembled by the peer.

6 Gateways

You can use different protocol connections between `vstt` hosts. Here is an example network using three different `vstt` tunnels:



In this scenario, Host2 and Host3 are `vstt` gateways. As two binaries on the same system would try to utilize the same FIFO files, you need to build `vstt2` (run `make vstt2`), which uses different FIFO files, as a second binary on the gateway and direct the output of `vstt` into the input of `vstt2`.

7 Comments, Feedback

Please send me feedback, typos, bug reports and requests to my ‘`steffenwendzel (at) gmx (dot) net`’ to make `vstt` better.