Steffen Wendzel[1]

vstt[2] documentation

[1] www.wendzel.de
[2] Very Strange Tunneling Tool

Contents

1 Disclaimer

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

2 Introduction

Network covert channels enable a stealthy transfer of information over a network. For an introduction, see my free online class on Github: `https://github.com/cdpxe/Network-Covert-Channels-A-U`

`vstt` is a tunneling tool (primarily for TCP connections). It can send your data via encapsulated in protocols.

I wrote `vstt` in 2006 as a second semester undergraduate student. It is far from perfect.

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: MacOS, 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. Similarily, `vstt` outputs data to a FIFO on the receiver-side 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 by reading from the output FIFO.

In case you want to use **TCP connections** without interacting with the FIFOs, you can use the tool `s2f` (short for *socket to FIFO*) that is shipped with `vstt`. `s2f` binds a socket to a FIFO.

4 Examples

Note: `vstt` prints 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 (ICMP tunnel and local FIFOs)

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

This setup requires different parameters to start `vstt`:

```
-p icmp      <- set the tunneling cover protocol to ICMP
-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 (as ICMP makes no use of ports) 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 recevier, 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*.

Finally, we just need to send the actual data that we want to transfer from the sender to the receiver.

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

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

If we now observe the output of the `reader` on the receiver, we will see the content of the sender's */etc/resolv.conf* that was transferred via ICMP.

4.2 Example 2 (tunneling a SSH connection)

Let's now use a SSH connection between two hosts over port 80 (e.g. because some firewall does not block HTTP but SSH). We use the protocol '`none`' because it is fast. '`none`' creates nothing but a plain TCP-based tunnel.

Note: You need **root** access to bind ports below 1024 on most Unix(-like) systems.

The setup works as follows: both systems start `vstt` to establish a tunnel they can communicate through. On the SSH server, we connect our `vstt`'s FIFO with the SSH service on port 22 (can be done using the `s2f` tool).

On the client machine, we also use the tool `s2f` (but in server mode, so that it accepts the SSH client's connection to forward the SSH packets through the tunnel). `s2f` communicates with the local `vstt` endpoint through its FIFO. Finally, we connect to the `s2f` port using our local SSH client.

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.

FIXME: check that whole setup once more!!!

On eygo, we start `vstt`. We receive configure it to receive data on port 80 and to send data to port 80 to the other `vstt`-endpoint.

```
eygo# ./vstt -p none -r 80 -t 80 -a 192.168.2.20 -m 192.168.2.21
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 a second terminal (also on eygo) we start `s2f`. It will listen on port 10003. We will connect to this port with the SSH client once the tunnel is established.

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

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

                # vstt -p none -r 80 -t 80 -a 192.168.2.20 -m 192.168.2.21 -c 100
```

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

On hikoki, we start `vstt` too:

```
hikoki# ./vstt -p none -r 80 -t 80 -a 192.168.2.21 -m 192.168.2.20
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 `vstt`-FIFOs to the local SSH-Server running on Port 22 via `s2f`:

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

IMPORTANT NOTE: You could alternativeley only start vstt one time without
                calling s2f:
                # ./vstt -p none -r 80 -t 80 -a 192.168.2.21 -m 192.168.2.20 -c 2
```

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

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

That's it. Your tunnel should be operational now.

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 POP3’s RETR-requests. If you want to hide your data a little bit: use POP3 (or ICMP).

5.3 ICMP

If all TCP ports are blocked, an ICMP tunnel (‘icmp’) could 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 larger packets from your applications within many small ICMP packets that will be re-assembled by the peer. In other words, vstt’s ICMP tunnel implements a simple version of a covert channel-internal control protocol[3].

6 Gateways

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

```
Client                                          Destination
  |                                                |
  ^                                                ^
Host1 |-----| Host2 |------| Host3 |----| Host4
     ICMP   ICMP    POP3   POP3    NONE NONE
```

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 via GitHub to enhance vstt.

[3] see S. Wendzel, J. Keller: Hidden and Under Control, in: Annales of Telecommunications, 2014. https://link.springer.com/article/10.1007/s12243-014-0423-x.