NAME

ssh-reverse-tunnel - create robust reverse ssh tunnels or VPNs

SYNOPSIS

Usage: ssh-reverse-tunnel client [options]
or: ssh-reverse-tunnel server [options]
or: ssh-reverse-tunnel **--help** or **--version** (gives the full manual or version info)

Without --vpn, a set of TCP port forwards is set using the --ports options.

With --vpn, a VPN is set up using PPP over SSH.

Identical commands are run on both client and server sides, with the exception of the first argument being "client" or "server" as appropriate.

OPTIONS

NOTE: All command-line options have analagous options available in the optional configuration file, which defaults to **/etc/ssh-reverse-tunnel.conf**.

Options common to all modes:

--client-mode-timeout *time*
Wait *time* seconds for a connection while in client mode

--configuration-file *file*
Use *file* as the configuration file instead of the default (**/etc/ssh-reverse-tunnel.conf**). If used, this option must be the first option after client or server.

--remote-commands
check for remote commands, see the REMOTE COMMAND FEATURE section

--server-host *server_host*
The server hostname from the client's point of view

--server-mode-timeout *time*
Wait *time* seconds for a connection while in server mode

--server-ssh-port *port*
The ssh port on the server (default is 22)

--server-user *server_user*
A user account on *server_host*

--ssh-extra-options *string*
Extra options string to be passed directly to ssh (remember shell quoting if *string* has spaces in it)

--verbose Give verbose output

ssh port forwarding option:

--ports *ports_string*
where *ports_string* is one or more of the pattern
server_port:localhost:*client_port* separated by spaces

where

server_port
: is the TCP port to use on the server side

client_port
: is the TCP port to use on the client side

For each pattern in *ports_string*, the *server_port* on the server is forwarded to the *client_port* on the client.

The same *ports_string* should be used on both client and server.

The *ports_string* patterns are actually passed directly to the ssh **-R** option. Note that "localhost" in the -R command refers to localhost on the client side. A way to remember this is that the host in an ssh port forward triplet is always relative to the port forward connection endpoint (as opposed to the ssh connection endpoint).

Server mode options:

--client-host *client_host*
The client hostname from the point of view of the client

--client-user *client_user*
A user account on *client_host*

--server-host-client-alias *host*
An alias for *server_host* on the server to avoid ssh host key conflict (see section: EXAMPLE - RESOLVING HOST KEY CONFLICTS)

VPN options:

--vpn Create a VPN tunnel over PPP instead of SSH port forwarding tunnel

--client-ssh-port *port*
The ssh port on the client (default is 22)

--client-vpn-address *client_address*
The client-side IP address will be *client_address*

--server-vpn-address *server_address*
The server-side IP address will be *server_address*

--client-pppd-command *client_pppd_command*
Execute *client_pppd_command* for pppd on client

--server-pppd-command *server_pppd_command*
Execute *server_pppd_command* for pppd on server

DESCRIPTION

ssh-reverse-tunnel allows you to set up tunnels so that you can connect to a computer which doesn't allow incoming connections. This is done by using a "server" computer which does accept incoming connections. The "client" which doesn't accept incoming connections connects to the server and sets up a reverse tunnel. There are two ways to do this.

Method 1: ssh reverse tunnels (without --vpn option)
: ssh's reverse tunnel feature (the **-R** option) can create reverse tunnels for specific TCP ports. The advantage of this is that you can set up a service, such as sshd or imapd, easily and connect from any host on the internet through a forwarded port on the server. The limitation is that only specified TCP ports are forwarded.

Method 2: VPN using PPP (with --vpn option)
: A VPN (virtual private network) can be created using ssh and pppd if you have root access or sudo access to pppd on both client and server. The advantage is that this is a full network interface with all the power and complexity that entails. Be sure and see the section EXAMPLE - CREATING A VPN for limitations with this approach.

ssh-reverse-tunnel is essentially a helper script, which will monitor the tunnel connection on both sides. When either the client or server side can't connect to the other side, it kills ssh and/or pppd on its side and sets up a marker to tell the other side to restart. When the connection goes down, it complains to standard output only once unless in verbose mode, so that the script can be run frequently by cron without filling up the logs when the connection is down. It also reports once after a connection has been re-established.

PREREQUISITES

ssh-reverse-tunnel will be impossible to use if you don't understand how to set up passwordless logins with ssh. One place this is described:
http://www.debian.org/devel/passwordlessssh
Google for "passwordless ssh" is also a good way to find out how to do this.

EXAMPLE - BASIC REVERSE PORT FORWARD

Suppose host *client_host* is behind a firewall that only allows outgoing connections, but you want to be able to ssh in to *client_host* from outside the firewall. To accomplish this, you will use *server_host*, which does allow incoming connections on port 2222, and forward port 2222 on *server_host* to port 22 on *client_host* using ssh reverse port forwarding. For this simple example, you only need a user account *client_user* on *client_host* and a user account *server_user* on *server_host*.

First, set up passwordless ssh logins from *client_user@client_host* to *server_user@server_host* and run:

```
client_user@client_host$ ssh-reverse-tunnel client \
        --ports 2222:localhost:22 \
        --server-user server_user --server-host server_host
```

You can now log in to *client_host* by connecting to *server_host* port 2222 locally:

```
server_user@server_host$ ssh -p2222 client_user@server_host
```

EXAMPLE - ADDING MORE PORT FORWARDS

The following extends the example so far to also forward *server_host* port 2225 to *client_host* port 25:

```
client_user@client_host$ ssh-reverse-tunnel client \
        --ports "2222:localhost:22 2225:localhost:25" \
        --server-user server_user --server-host server_host
```

EXAMPLE - RESOLVING HOST KEY CONFLICTS

There may be a problem with connecting to *server_host* port 2222 on *server_host*, though. If you have an ssh host key for *server_host* already, when you connect with the above command ssh will receive the host key from *client_host* because of the port forwarding. ssh will see that the host key for *server_host* doesn't match what it expects, and figure the key has changed because of a man in the middle attack, and will fail. One way around this is to create an alias *server_host_client_alias* to *server_host* in the **/etc/hosts** file on *server_host*, and then use the following to connect to *client_host*:

```
server_user@server_host$ ssh -p2222 client_user@server_host_client_alias
```

Another way that is less desirable, but may work if you don't have root access on *server_host* is to use the ssh option
"-oStrictHostKeyChecking=no". This will only work with passwordless login:

```
server_user@server_host$ ssh -p2222 -oStrictHostKeyChecking=no \
        client_user@server_host
```

EXAMPLE - CONNECT FROM ANYWHERE

If you want to be able to login to *client_host* from any host, you might expect that the ssh -g option would work, but it doesn't (see Debian Bug #228064 at http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=228064). Instead, add the line:

```
GatewayPorts yes
```

to the sshd configuration file (/etc/ssh/sshd_config on Debian) on *remote_host*. Be aware that this opens up ALL ports forwarded by sshd on *remote_host*, so if you don't want some other ports forwarded, you will need to set up your firewall appropriately, or complain about the -g bug.

You can now log in to *client_host* from any host with:

```
any_host$ ssh -p2222 client_user@client_alias
```

where client_alias is an alias for *remote_host* to solve the host key collision problem described in the previous section.

EXAMPLE - CREATING A ROBUST TUNNEL

If the simple example above meets your needs, you should probably evaluate the autossh program, described below in the SEE ALSO section. In my experience, when connection conditions are bad, the sshd process on the server which is holding the reverse-forwarded port open may need to be restarted. This currently requires root access on the server, see the LIMITATIONS AND BUGS section for the reason.

This script can be used to keep a tunnel up all the time if it is run regularly on both client and server via cron. When used this way, the first tunnel in *ports_string* should always be back to the ssh port on *client_host*, since this tunnel is used to test a round-trip ssh connection.

Continuing with the example, on *client_host*, cron should execute:

```
client_user@client_host$ ssh-reverse-tunnel client --ports 2222:localhost:22 \
        --server-user server_user --server-host server_host
```

and on *server_host*, cron should execute (as root):

```
root@server_host$ ssh-reverse-tunnel server --ports 2222:localhost:22 \
        --server-user server_user --server-host server_host \
        --client-user client_user --client-host client_host \
        --server-host-client-alias server_host_client_alias
```

Note that the second command is the same as the first with a few options added. The server command given would also work on the client side, with the first argument changed from "server" to "client".

To run this example using a configuration file, you would set up the values in the configuration file as above, then use the same configuration file on both sides. If all the options are set correctly in the configuration file, you would just run:

```
client_user@client_host$ ssh-reverse-runnel client
```
and

root@*server_host*$ ssh-reverse-tunnel server

Before setting up the cron scripts, the following should work without passwords:

client_user@client_host$ ssh *server_user@server_host*
root@*server_host*$ ssh -p2222 *client_user@server_host_client_alias*

EXAMPLE - CREATING A VPN

This is not the best way to create a VPN! It is however, the easiest way I know to create a semi-robust VPN to a computer behind a firewall. You don't need to change anything with your firewall or your NAT router, for instance.

There are several downsides to this method. It works poorly for large transfers or for connections where latency is important (like X, vnc, or shells). If the tunnel is restarted for whatever reason, any established connections will be dropped. See http://sites.inka.de/sites/bigred/devel/tcp-tcp.html for the technical discussion of why this is a bad way to implement a VPN. OpenVPN is a much better solution for a serious VPN that can be created for all situations where ssh-reverse-tunnel can be used and more. The only advantage of this script over OpenVPN for creating a VPN is that it's extremely easy to configure. Consider the VPN mode of the script a demo or toy.

For this example, we're going to create a VPN at IP 192.168.88.1 on *server_host* and IP 192.168.88.10 on *client_host*. ssh-reverse-tunnel will be run as root on both hosts every two minute via cron.

1) Setup passwordless login from root@*client_host* to *server_user@server_host*.

2) Set up sudo so that *server_user@server_host* can run pppd. Add the line:
server_user ALL=NOPASSWD: /usr/sbin/pppd
to /etc/sudoers on *server_host*.

3) Run this command on the client to start the tunnel, and later set it up to run via cron every 2 minutes:

root@*client_host*$ ssh-reverse-tunnel client --vpn \
--server-user *server_user* --server-host *server_host* \
--client-user *client_user* --client-host *client_host* \
--client-vpn-address 192.168.88.10 --server-vpn-address 192.168.88.1 \
--server-pppd-command "sudo pppd"

4) Set up passwordless ssh login from root@*server_host* to *client_user*@192.168.88.10.

5) Run the command in step 3 on the server every two minutes via cron, but change the first argument from "client" to "server".

MANUALLY RESTARTING THE TUNNEL

If you can't get a connection from *server_host* to *client_host*, create the file /home/*server_user*/.ssh-reverse-tunnel.restart.*client_host* on *server_host*. The next time ssh-reverse-tunnel is run for this tunnel on the client side, it will see this file and restart the ssh tunnel. The restart marker file will be deleted by the client side only after the tunnel is successfully restarted.

REMOTE COMMAND FEATURE

If the **--remote-commands** option is given in client mode, ssh-reverse-tunnel will look for the file /home/*server_user*/.ssh-reverse-tunnel.command.*client_host* on
server_host and execute the contents on *client_host* using /bin/sh. This is useful for debugging, and possibly other uses. The command file is deleted on *server_host* after it is executed.

WHAT'S GOING ON IN DETAIL

On the client side when run in client mode, the following occurs:

1) execute contents of the remote command file if needed (see REMOTE COMMAND FEATURE)

2) kill the client-side ssh tunnel process if the kill file exists (see MANUALLY RESTARTING THE TUNNEL)

3) if the tunnel is already running, return with exit status 0

4) if the tunnel is not running, start tunnel

a) if the tunnel is not running after *client_mode_timeout*, abort with error message

b) remove the kill marker file on *server_host* if it exists

On the server side when run in server mode as root, the following occurs:

1) attempt to create a connection to the client through the tunnel, then through this connection connect back to the server and create /home/*server_user*/.ssh-reverse-tunnel.server_connect.*client_host*

2) if the the marker file hasn't been created after *server_mode_timeout*:

a) kill the tunnel processes on the server

b) kill the client-side tunnel processes by creating the file on *server_host* described in the section MANUALLY RESTARTING THE TUNNEL

FILES

/etc/ssh-reverse-tunnel.conf

Default configuration file location. Command-line options override anything here.

LIMITATIONS AND BUGS

Only one tunnel from a given client to a server can exist, although one server can have tunnels from multiple clients. This shouldn't be too much of a problem since one tunnel can have multiple port forwards.

Server mode requires root access. This is because netstat **-lp** doesn't return the PID of port listeners to non-root users, even if those users own the process which is listening on the port. A Debian bug has been filed against this problem (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=292453).

ssh-reverse-tunnel may only work using the GNU utilities, since it does things like pull the 7th column from the output of netstat. I don't have the ambition to test on non-GNU utilities, although I will accept patches to make it more portable.

ssh-reverse-tunnel has only been tested with OpenSSH.

Report bugs to bugs@danielwebb.us.

AUTHOR

SEE ALSO

autossh - http://www.harding.motd.ca/autossh/

Use like a normal ssh command, except autossh keeps the connection up

VPN PPP-SSH HOWTO on http://www.linuxdoc.org

OpenVPN - http://openvpn.net

A full-featured VPN that can be used in all the places ssh-reverse-tunnel can and more