Setting Up a Reverse Tunnel Through a Firewall

The following howto was created when W1GHW, Gary, had a need to access an Allstar site that is located behind an unknown number of nat'ed/firewalled networks at the college where he is a professor. I suggested a reverse SSH tunnel and gave him some ideas how to achieve access to the system. He was able to make it work successfully as he has documented below.

The Concept

The need exists to log into a computer controlling an Allstar linked radio system when that system is behind a firewall with no knowledge of the address (IP) of the computer. Access is achieved through the establishment of a "reverse tunnel" back to the radio's computer through an SSH session initiated by that computer to a remote system located anywhere on the outside of the firewall.

The setup invokes a script either at boot, by a cron or from a console session that establishes the tunnel. The script is also designed to periodically check to determine if the SSH session is still running and re-establish it if it is not. In addition, the SSH shell has an internal feature that monitors itself to insure that the session does not close due to lack of activity since the session will be mostly dormant and firewalls tend to try to clean up connections that are dormant for long periods of time.

In the Client-Server model, the client is the computer that initiates an SSH session, in this case at the radio site behind the firewall, and the server is the remote computer that receives the request for an SSH session located somewhere on the outside of the firewall. To create the "reverse tunnel" the client initiates the SSH shell session with the server using the –R switch that instructs the server to open a free port and listen for activity. If a separate login to the server computer requests another SSH session through that port, one will be opened back to the client, localhost, using the port specified during setup.

Below are some of the files involved in the setup with the key variables set to values that have worked. These may or may not be the only settings that work, as an exhausted study of all possible configurations has not been undertaken.

Server Side Variables

File: /etc/ssh/ssh_config located in on server
#ServerAliveInterval 25
#ServerAliveCountMax 2

Notes: If these two variables exist in the file, they should be commented out with the "#" character at the beginning of the line so they are not active.

File: /etc/ssh/sshd_config located in on server
TCPKeepAlive no
#ClientAliveInterval 30
#ClientAliveCountMax 3

Notes: There is no need for the TCPKeepAlive feature to be engaged. The other two should be commented out if they are in the file.

Client Side Variables

File: ~/.ssh/config located off of the root directory on client as root (note the "." before "ssh")
KeepAlive yes
ServerAliveInterval 60

Notes: These two variables are **key** to getting the tunnel to remain active when there are long periods of idle traffic. They are placed in a file called config in the .ssh directory off of root. They also might work if put in the /etc/ssh/ssh_config file, but that has not been tested.

File: /etc/ssh/sshd_config located on client as root
#ClientAliveInterval 30
#ClientAliveCountMax 3

Notes: These two variables if in the file should be commented out so they are not active.

Client Side Scripts

File: Tunnel.sh located on client as root. (Any filename will do)

```
#!/bin/bash
# createTunnel() appears first in this script but is not executed until called from
   code below. This procedure is the code that actually creates the tunnel.

createTunnel() {
  /usr/bin/ssh -N -f -R x:localhost:y username@server_address -p ssh_port

#  The –N switch means no console will be created. The –f switch runs the process
   in the back ground. The –R switch sets up the reverse tunnel back to the client.
   It instructs the server to open port x and listen for traffic. If that traffic
   wants to connect back to the client (known as "localhost") it should use port y to
   establish an SSH session with the client. "username@server_address" is the
   account on the server that the client logs into when the SSH session is started.
   To avoid the system hanging because it wants a password (but not console
   session to get it), Public Key Authentication must be used. (See below)

  echo $(pidof ssh) > stored_pid       # Collect and store the PID for the SSH
session.
  if [[ $? -eq 0 ]]; then              # Check to see if the session was established
    echo Tunnel created successfully
  else
    echo An error occurred creating a tunnel. RC was $?
  fi
}

kill -0 $(pidof ssh) 2> /dev/null      # Check to see if there is a PID for an SSH
session
# or use "kill -0 $(cat stored_pid) 2> /dev/null".  If there are multiple SSH
       sessions open, this will focus on the PID of the session established for the
       tunnel. It uses the PID saved in the file "stored_pid" when the tunnel was
       set up. Also, the kill command in this form puts out a usage message as an
       error so it is redirected to the null device.

if [[ $? -ne 0 ]]; then
  echo "Job not running. Creating new tunnel connection"
  createTunnel
```

```
else
  echo "Job running."
fi
#End of SCRIPT
```

Notes: PublicKeyAuthentication: See https://www.tunnelsup.com/ssh-without-password/. This link discusses how to set up a public key for automatic authentication to avoid a password dialogue when no console session is established.

Example: If the server computer uses port 22 as the default for an SSH session, if I am using port 2222 on the server computer with the account root@billyboy.com to establish the reverse tunnel and if the client computer uses port 222 as the default for any SSH session, the command would be: ssh –N –f –R 2222:localhost:222 root@billyboy.com –p 22.

File: retun.sh located on client as root. This is an abbreviated script that establishes the SSH session on boot. (See "on boot" below.)

```
#!/bin/bash
#
kill -0 $(pidof ssh) 2> /dev/null  # Check to see if there is a running SSH. If so,
                                   kill it. If multiple sessions are open, this might kill
                                   them all.
if [[ $? -eq 0 ]]; then
  kill -9 $(pidof ssh) 2> /dev/null
fi
/usr/bin/ssh -N -f -R x:localhost:y username@server_address –p ssh_port   #
Setup the tunnel.
#End of SCRIPT
```

On Boot: *~/scripts/retun.sh* # Entry in /etc/rc.local located on the client and put after the asterisk startup script.

Cron entry on client

```
*/15 * * * * ~/tunnel.sh > tunnel.log 2>&1  # Checks every 15 min. to see if tunnel is
                                             still active and re-establishes it if it has
                                             terminated.
```