

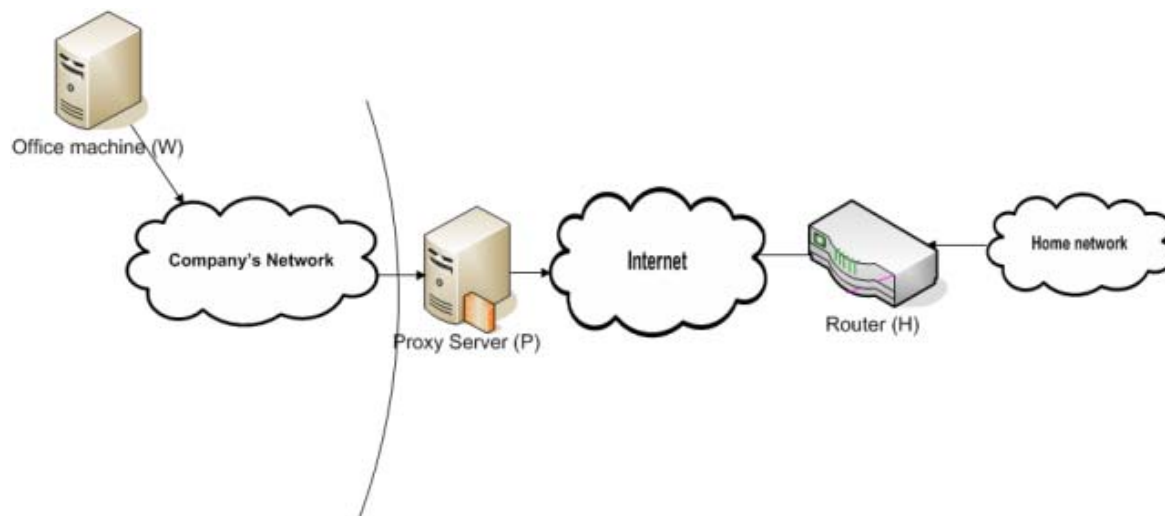
Tunneling Your Company's Firewall For Fun and Profit

How-to Get Into Your Office Computer From Home (..without using VPN)

Overview

Yes, it's possible. Let me first give you an overview of the setup. You work with a company 'XYZ'. At office, you cannot access internet directly and you 'browse' internet using HTTP(S) proxy. Back at home, you have an internet connection. You want to access the office computer from home, but you don't have the VPN access. Well, don't despair. You can use 'ssh tunneling' with some cool tricks to solve your problem.

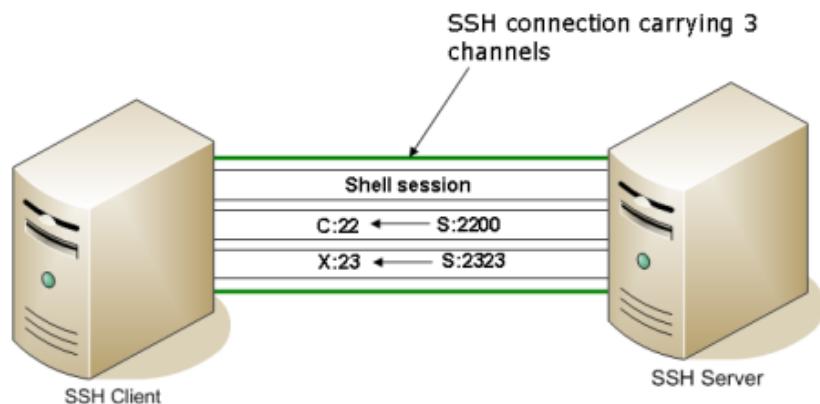
Figure 1. setup



SSH Tunneling

It's not something new and is well known to most of the computer guys. So, I'll discuss it briefly. SSH protocol has the concept of channels within an ssh connection. What it means is that you can have more than one communication channel within one ssh connection. This is called tcp port forwarding in ssh realm.

Figure 2. ssh tunneling



You can setup tcp port forwards in any direction independent of ssh connection direction. What it means is that if we have a connection (C - client, S - server):

C:17315 ---> S:22

Within this connection, we can have tcp forwardings like following:

S:2200 ---> C:22 (Remote forward, remote to local machine)
S:2323 --> X:23 (Remote forward, remote to some other machine X on the client side)
C:25 ---> S:25 (Local forward)

And all these connection remain hidden inside the main ssh connection. Of course, these capabilities can be limited by configuring ssh server.

This is what we are going to use to solve our problem. SSH server will run on a machine at your home and SSH client will run on an office machine. But establishing this connection is not easy as you are not directly connected to the internet. Assuming that you cannot do anything with the firewall, which is true for most of the employees, you are left with only one option. That is to punch holes in you HTTP(S) proxy.

To make sure that you have what you need, let's start with requirements.

Requirements

1. A machine at your workplace. You will need to have access to this machine from within the office LAN. I am going to assume a linux server. It can be a Solaris or Windows machine too. Only setup will be a little different for windows.

Office machine, W - 172.17.78.121

2. An HTTP(S) proxy. You just need the ip address and port. Tip: Find it from your browser.

HTTP(S) proxy, P - http-proxy.xyz.com:8080 (A Web proxy)

3. A machine at your home. This machine should be connected to the internet, with port 443 accessible from outside. You don't have any? Look around. Chances are that you have one. My cute little WRT54G with OpenWRT firmware works fine for this purpose. In fact, you can use any router which runs an ssh server and gives you an option to modify it. There are not many such routers though ;-). Since, router is always connected to internet, it makes a perfect choice for this machine. I am going to demonstrate this setup using Wrt54G running OpenWRT firmware.

Home machine, H - 67.167.12.78 (Linksys WRT54G running OpenWrt)

Tip: If your ip address keeps changing, you can use free dynamic dns service from dyndns.org. They will let you call your machine by a fixed hostname something like: *myhost.dyndns.org*.

From the discussion above, we know that we are going to use ssh tunneling over an http(s) proxy as a solution to our problem. To start with, let's look at the process of punching holes in http(s) proxy:

Punching Hole in HTTP(S) Proxy

To understand it let's dwell upon the working of HTTP(S) proxies a bit.

HTTPS is secure http. What it means is that data is all encrypted between browser and web server, so nobody can peep at your data on the way. Not even your proxy server.

To connect to an https site, your browser connects to http(s) proxy at some predefined port (generally 80 or 8080) and sends a "CONNECT webserver:443 HTTP/1.1" request to the proxy server. Proxy server establishes a TCP connection with the web server and returns a "HTTP/1.1 200 Connection Established" message. Once it is done, proxy server's role gets limited to just forwarding the packets between your browser and web server. Your browser starts a TLS session with the web server and transfers encrypted data thereafter. So, your proxy doesn't even know what your browser is doing after that.

SSH Connection

So, http(s) proxy doesn't come to know what's going on between you browser and the web server. Your browser might even be talking to an ssh server at the other end. But, we know browsers don't do such things ;-). But some ssh clients are smart enough to pose to proxy servers as http(s) clients. Putty is one of them. Of course, for these clients to be able to piggyback on HTTP(S) proxy, you will have to run ssh server at port 443 as most of the http(s) proxies allow "CONNECT" only at port 443.

However, there are some ssh clients which don't support http(s) proxy. OpenSSH is one of them. If you use that as I do, you need to use some other software to talk to http(s) proxy on behalf of our client. You can use proxytunnel available at <http://proxytunnel.sourceforge.net>. To use proxytunnel, you need to specify following option for OpenSSH ssh:

```
ProxyCommand $HOME/tools/proxytunnel -g http-proxy.xyz.com -G 8080 -d %h -D 443

-g gateway (your http(s) proxy)
-G gateway port (http(s) proxy port - 8080)
-d destination host (%h will be substituted by the host name to connect)
-D destination port (I have kept it fixed. You can keep it as %p to take it from ssh)
```

This tells ssh to use this command to connect to our host. You can put it in `~/.ssh/config`, default configuration file for OpenSSH client. However, to avoid letting it affect other ssh sessions, you would probably want to keep it in a separate file `~/tunnel/config`. You can tell ssh client to use this file, by specifying `-F configfilepath` option at command line.

At your home machine H, make your ssh server listen on port 443 and configure the firewall to allow connections on port 443 from the outer world. OpenWRT uses dropbear as ssh server which is a tiny implementation of SSH protocol. You can make dropbear listen at 22 (for general use) and 443 by using following command-

```
root@OpenWrt:~# /usr/bin/dropbear -p 22 -p 443
```

Now you are all set to establish a connection from W to H. Just issue the following command:

```
[you@Office ~] ssh -F ~/tunnel/config -l root myhost.dyndns.org
```

Quieting SSH connection

We are going to use this ssh connection only for the tunneling purpose, i.e. no shell. So, it's a good idea to quiet it. We'll use public key based authentication, so that ssh doesn't ask for the password. To do that just generate a public/private key pair using `ssh-keygen` that comes with OpenSSH. Keep the private key on the office machine (W) and paste the content of the public key in `~/.ssh/authorized_keys` file on your OpenWRT(H). Let's say that the private key file is `router_priv`. Then you can call ssh in the following way for password less authentication:

```
[you@Office ~] ssh -F ~/tunnel/config -l root -i router_priv myhost.dyndns.org
```

To quiet it further, use it as following:

```
[you@Office ~] ssh -n -F ~/tunnel/config -l root -i router_priv \
myhost.dyndns.org >/dev/null 2>&1 &
```

`-n` redirects stdin to `/dev/null`.

Keep the connection going

Well, http(s) proxies have a behavior of dropping the connection after some time interval (approx. 120 sec), which is quite reasonable as they expect it to be an http connection. To stop proxies from doing that, we need to keep transferring something on the connection. To do that, you can write a small script on your router (H). Only task of this script is to keep printing something at an interval of 60 seconds.

```
root@OpenWrt:~# cat keepalive
while true; do echo "Keep this alive"; sleep 60; done
```

Now call ssh from the Office machine in the following way:

```
[you@Office ~] ssh -n -F ~/tunnel/config -l root -i router_priv \
myhost.dyndns.org ./keepalive >/dev/null 2>&1 &
```

Here you go. You have a very quiet and long-lasting ssh connection between W and H. But, of course, this connection is of no use until unless we add a hidden channel to it.

Port Forwarding

To put your ssh connection to any use, you need to use port forwarding. Say, you want to forward port 2200 on H to port 22 on W, so that when you connect to port 2200 of H, you are automatically connected to port 22 of W. You should request this port forwarding while setting up the ssh connection between the two machines:

```
[you@Office ~] ssh -n -R 2200:localhost:22 -F ~/tunnel/config -l root \
-i router_priv myhost.dyndns.org ./keepalive >/dev/null 2>&1 &

-R 2200:localhost:22 -- "Forward traffic on port 2200 of remote machine to port 22 of loc
```

That would be it. But, there is a small annoyance. You cannot login to the router directly at port 2200. You have to first get into the router either by sshing at port 22 or telnet and then issue the following command from within the router to get connected to the office machine W:

```
root@OpenWrt:~# ssh -l you -p 2200 localhost
```

The reason that the direct login to remote forwarded port doesn't work is - "dropbear binds remote port forwardings to the loopback address. This prevents other remote hosts from connecting to forwarded ports."

To fix it, you can recompile dropbear for your router.

Recompiling dropbear for OpenWRT

To recompile dropbear for your router, you need 'build toolchain' for your router platform. For WRT54G it is mipsel. You can get mipsel build toolchain for x86 host from the following link-

<http://downloads.openwrt.org/people/nbd/whiterussian/OpenWrt-SDK-Linux-i686-1.tar.bz2>

Download and extract this file on a linux machine. Also download dropbear source package and follow the instructions below to compile dropbear-

1. Set path variable to include path to build toolchain utilities:

```
export PATH=$PATH:~/OpenWrt-SDK-Linux-i686-1/staging_dir_mipsel/bin
```

2. cd into dropbear source directory and issue following command to configure dropbear -

```
./configure --build=i686-pc-linux-gnu --host=mipsel-linux \
--prefix=$HOME --disable-zlib --disable-lastlog
```

3. Edit options.h (not necessary in many cases)

```
#define DROPBEAR_RANDOM_DEV "/dev/random"
to
#define DROPBEAR_RANDOM_DEV "/dev/urandom"
```

4. Edit tcp-accept.c

```
98c98
```

```
<  nsocks = dropbear_listen("", portstring, socks,
---
>  nsocks = dropbear_listen(NULL, portstring, socks,
```

5. Compile

```
make PROGRAMS="dropbear" STATIC=1
```

6. Strip

```
mipsel-linux-strip dropbear
```

Transfer this compiled dropbear to your router. You can use wget or scp available on OpenWRT to transfer this file. Test this binary by running it on some other port. Once you are sure that it's running fine, replace original one with this binary.

Now you are all set. If your router's internal ip address is 192.168.1.1, you can login to your office machine W by calling putty with following options:

```
putty -l you -P 2200 192.168.1.1
```

Isn't it cool? :-) Well, let me know.

Putting it all in scripts

To avoid the pain of typing such a complex command everytime, you can put it in scripts. Here is how you could set it up:

```
[you@Office ~]$ ls tunnel
config  mktun  proxytunnel  router_priv

[you@Office ~]$ cd tunnel

[you@Office tunnel]$ cat config
ProxyCommand ./proxytunnel -g http-proxy.xyz.com -G 8080 -d %h -D 443

[you@Office tunnel]$ cat mktun
#!/bin/sh
cd `dirname $0`
ssh -F config -n -R2200:localhost:22 -l root -i router_priv \
myhost.dyndns.org ./keepalive >/dev/null 2>&1 &
```

To make sure that this tunnel always remains up, you can set up a cronjob to run a script like this every 5 min or so:

```
[you@Office ~]$ cat startTun
#!/bin/sh

if pgrep proxytunnel
then
    echo "Running"
else
    echo "Not running. Starting now"
    /home/you/tunnel/mktun
fi
```

Disclaimer

I am sure, by now you can realize all the possible bad uses of this technique. Chances are that your company wouldn't like it. So, don't use it or use it knowing the risk. Don't hold me responsible for whatever happens to you once you are caught ;-)

Sysadmins and network admins, please don't hate me. Better secure your proxy servers and network. ;-)

References

- More information on OpenWRT can be found from their website: <http://openwrt.org>
- Google on SSH port forwarding: <http://www.google.com/search?q=ssh+port+forwarding>
- Proxytunnel can be downloaded from following location: <http://proxytunnel.sourceforge.net>
- Dynamic DNS service homepage: <http://www.dyndns.org>
- Mipsel build toolchain can be downloaded from here:
<http://downloads.openwrt.org/people/nbd/whiterussian/OpenWrt-SDK-Linux-i686-1.tar.bz2>

This link seems to be little unstable. If you cannot find it here, search on google or contact me.

- Dropbear homepage: <http://matt.ucc.asn.au/dropbear/dropbear.html>
- OpenSSH homepage: <http://www.openssh.com/>

About Me

I am a unix sysadmin by profession. I like to find out how things work. Since I know computers, generally I end up finding things about computers. My blog is an attempt to share with you guys whatever I learn.

Manu Garg

manugarg at gmail dot com

<http://www.manugarg.com>