HowTo ASIX Tunnel SSH

Curs 2015-2016

Objectius

---

1. Túnel SSH directe
   a. El model de funcionament dels Túnels SSH
   b. Túnel SSH directe host to destí (simple)
   c. Túnel SSH directe host to destí to remote

2. Túnel SSH Reverse
   a. El mpdel de funcionament dels túnels Reverse.
   b. Túnel reverse host to destí (simple).

   c. Túnel reverse host to destí to remote.

3. Exemples implementats amb:
   a. Accés web
   b. Accés serveis xinetd: daytime, echo.
   c. Accés a SSH.
   d. Accés PostgreSQL.

Documentació

Aquest document ha estat elaborant utilitzant com a eina de treball un sistema GNU/Linux Fedora 20.

- Documentació de les pàgines man de les ordres.
- Fedora Documentation, Fedora 21. Sistem Administration Guide: 7. OpenSSH

- Observar els gràfics descriptius de les situacions de: https://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/

- http://ubuntuguide.org/wiki/Using_SSH_to_Port_Forward
- https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding

Ordres:

ssh	ssh-add	ssh-agent	ssh-copy-id	sshd	sshd-keygen
sshfs	ssh-keygen	ssh-keyscan			

Tunnel SSH

---

man ssh

TCP FORWARDING
Forwarding of arbitrary TCP connections over the secure channel can be specified either on the command line or in a configuration file. One possible application of TCP forwarding is a secure connection to a mail server; another is going through firewalls.

In the example below, we look at encrypting communication between an IRC client and server, even though the IRC server does not directly support encrypted communications. This works as follows: the user connects to the remote host using ssh, specifying a port to be used to forward connections to the remote server. After that it is possible to start the service which is to be encrypted on the client machine, connecting to the same local port, and ssh will encrypt and forward the connection.

The following example tunnels an IRC session from client machine “127.0.0.1” (localhost) to remote server “server.example.com”:

```
$ ssh -f -L 1234:localhost:6667 server.example.com sleep 10
$ irc -c '#users' -p 1234 pinky 127.0.0.1
```

This tunnels a connection to IRC server “server.example.com”, joining channel “#users”, nickname “pinky”, using port 1234. It doesn't matter which port is used, as long as it's greater than 1023 (remember, only root can open sockets on privileged ports) and doesn't conflict with any ports already in use. The connection is forwarded to port 6667 on the remote server, since that's the standard port for IRC services.

The -f option backgrounds ssh and the remote command “sleep 10” is specified to allow an amount of time (10 seconds, in the example) to start the service which is to be tunnelled. If no connections are made within the time specified, ssh will exit.

-L [bind_address:]localport:hostremote:hostremoteport user@sshserver

Specifies that the given port on the local (client) host is to be forwarded to the given host and port on the remote side. This works by allocating a socket to listen to port on the local side, optionally bound to the specified bind_address. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to host port hostport from the remote machine. Port forwardings can also be specified in the configuration file. IPv6 addresses can be specified by enclosing the address in square brackets. Only the superuser can forward privileged ports.

By default, the local port is bound in accordance with the GatewayPorts setting. However, an explicit bind_address may be used to bind the connection to a specific address. The bind_address of “localhost” indicates that the listening port be bound for local use only, while an empty address or ‘*’ indicates that the port should be available from all interfaces.

X11 FORWARDING

If the ForwardX11 variable is set to “yes” (or see the description of the -X, -x, and -Y options above) and the user is using X11 (the DISPLAY environment variable is set), the connection to the X11 display is automatically forwarded to the remote side in such a way that any X11 programs started from the shell (or command) will go through the encrypted channel, and the connection to the real X server will be made from the local machine. The user should not manually set DISPLAY. Forwarding of X11 connections can be configured on the command line or in configuration files.

The DISPLAY value set by ssh will point to the server machine, but with a display number greater than zero. This is normal, and happens because ssh creates a “proxy” X server on the server machine for forwarding the connections over the encrypted channel.

ssh will also automatically set up Xauthority data on the server machine. For this purpose, it will generate a random authorization cookie, store it in Xauthority on the server, and

verify that any forwarded connections carry this cookie and replace it by the real cookie when the connection is opened. The real authentication cookie is never sent to the server machine (and no cookies are sent in the plain).

If the ForwardAgent variable is set to “yes” (or see the description of the -A and -a options above) and the user is using an authentication agent, the connection to the agent is automatically forwarded to the remote side.

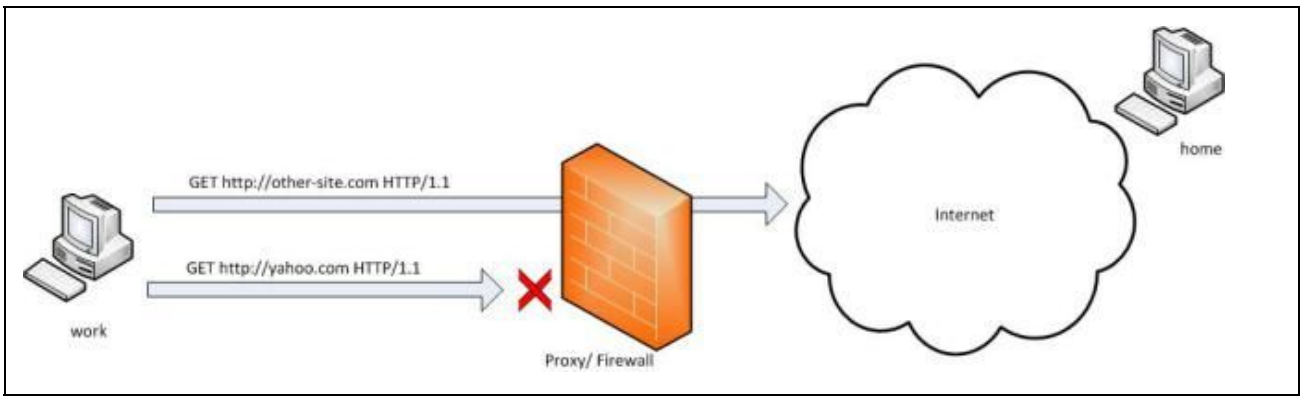
Exemples documentació externa

Exemples extrets del document “SSH Tunneling Explained” de la web “https://chamibuddhika.wordpress.com ”.

Per poder realitzar túnels SSH, tant directes com reverse, cal des d’un host poder realitzar una connexió ssh a un altre host, on evidentment hi ha engegat un servidor ssh.

Exemple 1:

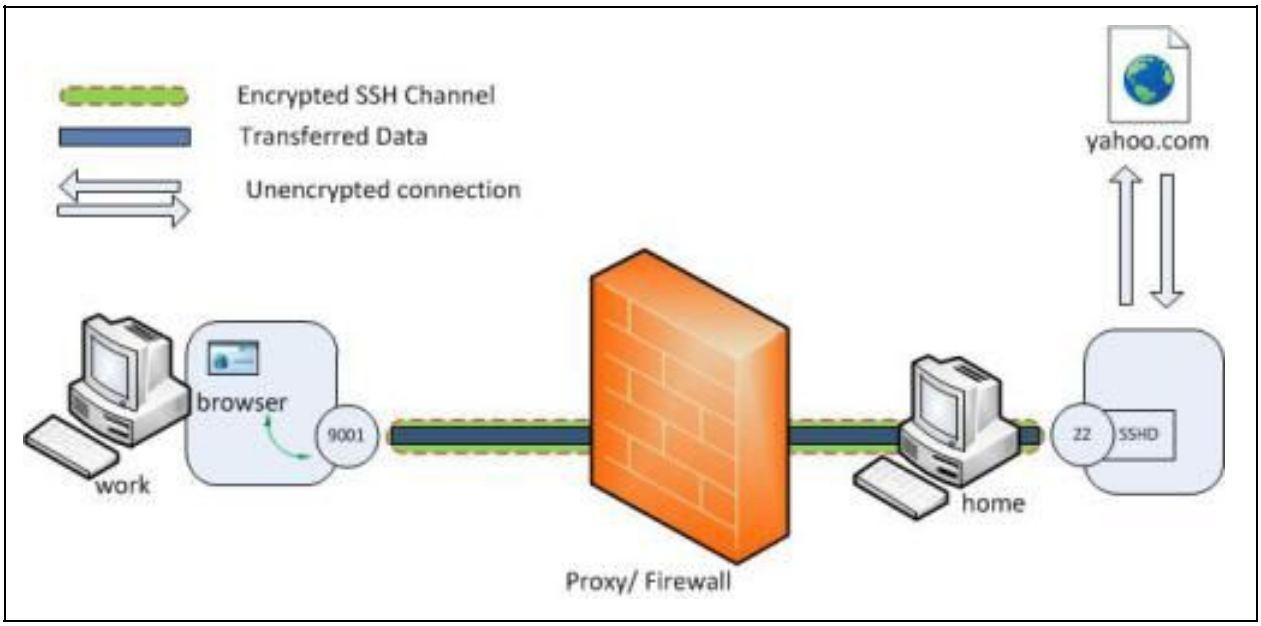
Crear un túnel del host work al host home, fent que aquest es connecti al port 80 de yahoo.com.



[work]$ ssh -L 9001:yahoo.com:80 home

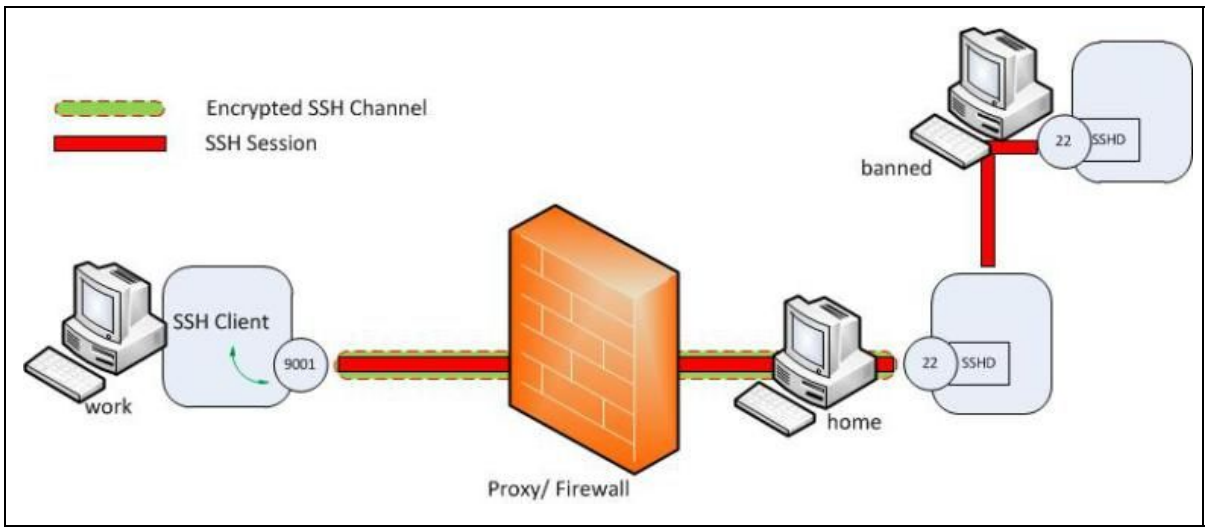
D’aquesta manera accedint al port local 9001 del host work en realitat s’accedeix al port 80 de yahoo.com, a través de home:

[work]$ telnet localhost 9001
GET / HTTP/1.0



Exemple 2:

En aquest exemple es realitza una connexió del ssh de work al ssh del host banned, a través del host home, ja que el firewall no permet l'accés directe de work a banned.



[work]$ ssh -L 9001:banned:22 work@home

Ara des del host work si es realitza una sessió ssh al propi port 9001 en realitat s'està accedint al ssh del host banned.

[work]$ ssh -p 9001 localhost

Exemples @edt

Exemple 3:

client ------> desti (ssh-server, web-server)

Establir una connexió al port 80 del host destí, fent passar el tràfic insegur HTTP per el túnel SSH . En el host client s'obrirà un port local 3030 que portarà el tràfic HHTP per dins del túnel fins al port 22 del host destí. En el destí el dimoni SSH encaminarà aquest tràfic al port 80 del propi destí.

En el destí CAL que s'executi el servei SSHD (el dimoni) i cal que l'usuari de client pugui establir una sessió SSH al destí.

[client]$ SSH -L 3030:localhost:80 user@desti

Ara en el host client accedint al port 80 local en realitat s'està accedint via el túnel SSH al port 80 del host destí.

[client]$ telnet localhost 3030

****atenció**** localhost no significa el host client sinó el host destí!

Exemple 4:

Ídem anterior fent un túnel SSH per accedir al servei daytime (del xinetd) del host destí. En aquest exemple s'utilitza -f per deixar el túnel SSH en background, executant la comanda sleep. Així no consumeix la consola.

[client]$ SSH -f -L 3031:localhost:13 user@desti sleep 100000
[client]$ telnet localhost 3031

Exemple 5:

client -- -- -- -- (firewalll) -- -- --- --- --- --> (80) destí
(3032) web server
|------------------> (22)SSH-Server ----------(80)
No es pot realitzar una connexió al port 80 del destí (un web server) perquè el firewall ho impedeix. Si es pot accedir al servidor SSH del SSH-Server. I aquest host (el SSH-Server) si que pot creuar el firewall i accedir al host destí al servei web (o al que sigui!).

El túnel SSH consisteix en establir un port local 3032 al host client que connecta via ssh al host SSH-Server (creuant la zona pública o insegura). El servidor SSH genera una connexió

al port 80 del destí, és tràfic HHTP insegur, però dins del firewall (que suposem una zona segura).

Per exemple el client és un host en un locutori internet, la connexió SSH és fins al router/firewalll de casa o de l'empresa, que té un servei SSH actiu i on el client es pot connectar. Des d'quest servidor SSH s'estableix una connexió normal a un host de dins de casa o de l'empresa (on el tràfic suposem que és segur!).

[client]$ SSH -L 3032:destí:80 user@ssh-server
[client]$ telnet localhost 3032
GET / HTTP/1.0

Observar que 3032 és el port local del client. desti:80 indica el host final al que contactar, a quin host i port el servei SSH reenviarà el que rep de client. user@ssh-server indica a quin host s'estableix el túnel, el tràfic xifrat. Usualment és un host que fa de frontera entre una zona insegura i una zona segura (que pot creuar el firewall) o que fa de router entra una xarxa pública i una de privada.

Exemple 6:

Ídem exemple anterior connectant a un host de classe al servei daytime. Client host i25, servidor SSH host i26 i servei al que accedir (en aquest cas daytime) al host i27:

[i25]$ SSH -f -L 3033:i27:13 user@i26 sleep 100000
[i25]$ telnet localhost 3033

Exemple 7:

Ídem demostrant que és el servidor SSH el que fa la resolució de noms del host destí al que accedir.

- Imaginem un host client en un locutori d'internet (host i26),
- Connectem al router/SSH-Server de casa que té el port SSH obert per establir el túnel (host i26), és una conexió de la xarxa internet, IPs públiques,
- Accedir a un host de dins de casa de la xarxa privada local de casa, a un host anomenat serverJocs (host i27/serverjocs). Aquest nom de host només és vàlid dins de la xarxa local.

Primerament afegir al servidor ssh una entrada al /etc/hosts anomenada serverJocs que identifiqui el host destí (host i27)
[i26]# echo “”192.168.2.57 serverjocs”” >> /etc/hosts

Establir el túnel des del host client (host i25) a un destí que és un nom de host de la xarxa privada local interna de casa (i que evidentment no es resoluble des del locutori on ens estem connectant)

[i25]$ SSH -L 3034:serverjocs:13 user@i26
[i25]$ telnet localhost 3034

Reverse Tunnel

Remote Port Forwarding

Remote port forwarding lets you connect from the *remote* SSH server to another server. To use remote port forwarding, you need to know your destination server, and two port numbers. You should already know your destination server, and for basic uses of port forwarding, you can usually use the port numbers in Wikipedia's list of TCP and UDP port numbers.

For example, say you wanted to let a friend access your remote desktop, using the command-line SSH client. You would use port number 5900 (the first *VNC* port), and destination server *localhost*:

```
ssh -R 5900:localhost:5900 guest@joes-pc
```

The -R option specifies *remote* port forwarding. For the duration of the SSH session, Joe would be able to access your desktop by connecting a VNC client to port 5900 on his computer (if you had set up a shared desktop).

-R [bind_address:]localport:hostremote:hostremoteport user@sshserver

Specifies that the given port on the remote (server) host is to be forwarded to the given host and port on the local side. This works by allocating a socket to listen to port on the remote side, and whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to host port hostport from the local machine.

Port forwardings can also be specified in the configuration file. Privileged ports can be forwarded only when logging in as root on the remote machine. IPv6 addresses can be specified by enclosing the address in square braces.

By default, the listening socket on the server will be bound to the loopback interface only. This may be overridden by specifying a bind_address. An empty bind_address, or the address '*', indicates that the remote socket should listen on all interfaces. Specifying a remote bind_address will only succeed if the server's GatewayPorts option is enabled (see sshd_config(5)).

If the port argument is '0', the listen port will be dynamically allocated on the server and reported to the client at run time. When used together with -O forward the allocated port will be printed to the standard output.

Exemples Reverse Tunnel

Exemples extrets del document "SSH Tunneling Explained" de la web "https://chamibuddhika.wordpress.com ".

Per poder realitzar túnels SSH, tant directes com reverse, cal des d'un host poder realitzar una connexió ssh a un altre host, on evidentment hi ha engegat un servidor ssh.

Exemple 1:

Des de dins d'una organització o xarxa local s'estableix un túnel ssh cap a un altre host per obrir-hi en aquest client extern una 'porta' per on accedir a aquesta xarxa interna/local. Usualment els firewalls filtren el tràfic d'entrada però permeten tràfic de sortida. Per exemple usualment està filtrat el tràfic SSH entrant però es permet el tràfic SSH sortint.

Establir un túnel invers des del host de treball de casa (host i27) a un host extern públic (host i26). El host de casa (host i27) té engegat el servei daytime (port 13). A aquest servei daytime s'hi pot accedir des de dins de la xarxa interna de casa, però no de fora. Amb un túnel reverse ssh es deixarà una porta oberta perquè des de l'exterior es pugui accedir a un servei (el daytime) de la xarxa interna privada de casa. Tot el que cal és permetre el tràfic ssh.

[i27]$ SSH -R 3035:localhost:13 user@i26

Així des del host privat (host i27) s'estableix un túnel cap enfora al host públic (host i26). En aquest host i26 es crea un port 3035 que permet accedir al servei daytime, port 13 del host privat i27. En connectar a aquest port 3035 local del i26 s'obté l'hora proporcionada per el servei daytime del host i27. Sense el túnel reverse el firewall de la xarxa local de casa no permetria l'accés al servei daytime del i27.

[i26]$ telnet localhost 3035

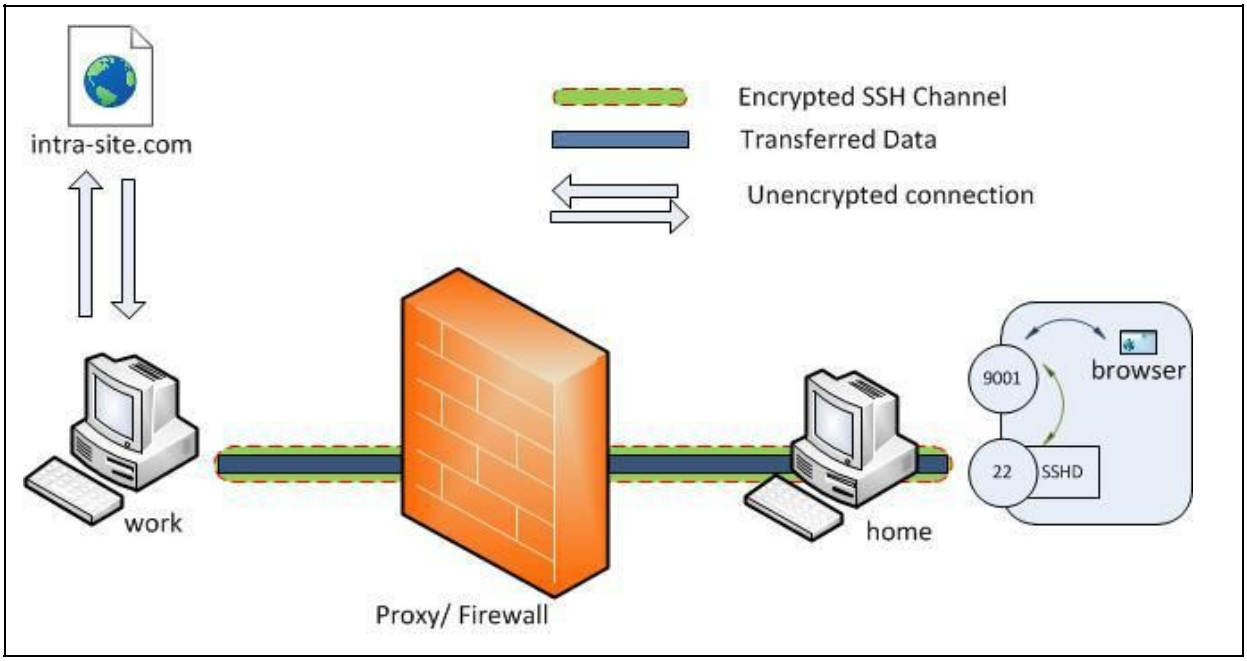
Exemple 2:

En aquest exemple des del host work (dins de la zona protegida del firewall) es crea un túnel invers per permetre al host home accedir al servei de intranet, que no es accessible des de fora perquè el firewall no ho permet.

En el host home es pot accedir localment al port 9001 i en realitat s'està accedint a la intranet de dins del firewalll.

[work]$ ssh -R 9001:intra-site.com:80 user@home

[home]$ telnet localhost 9001



Exemple 3:

Implementar a l'aula un túnel invers per permetre a un client accedir a un servei que està en una zona protegida o en una xarxa privada. és a dir, el client en principi no té accés al servei que proporciona el host destí. Però el host *client SSH* si que té accés als serveis del host destí.

- En una intranet/xarxa-privada hi ha un host (host i27) que proporciona un servei daytime (del xinetd). Aquest servei nomes és accessible per la intranet.
- Un host de la intranet (host i26) fa la funció firewall/router o de intermediari (on està l'usuari) estableix un túnel ssh invers amb el client.
- El host client (host i25) disposarà del port local 3036 que li permetrà l'accés passant pel túnel ssh al servei daytime del host destí, accés que no podria realitzar directament fora del túnel.

Primerament engegar el servei daytime-stream del xinetd en el host destí (host i27) i configurar l'accés a aquest servei únicament per a ell mateix i per al host intermediari (i26), amb la directiva:

only_from = localhost 192.168.2.57 192.168.2.56

Establir des del host intermediari (host i26) el túnel al client (host i25):

[i26]$ SSH -R 3036:i27:13 user@i25

Observem que a l'ordre a part de tenir el -R els ports s'indiquen com usualment, però signifiquen 'el revés'. El port 3036 s'obrirà en el client (user@i25) i connectarà al port 13 del host destí i27.

Dynamic Port Forwarding

Dynamic Port Forwarding

Dynamic port forwarding turns your SSH client into a *SOCKS proxy server*. *SOCKS* is a little-known but widely-implemented protocol for programs to request any Internet connection through a *proxy server*. Each program that uses the proxy server needs to be configured specifically, and reconfigured when you stop using the proxy server.

For example, say you wanted Firefox to connect to every web page through your SSH server. First you would use dynamic port forwarding with the default SOCKS port:
ssh -C -D 1080 laptop

The -D option specifies *dynamic* port forwarding. 1080 is the standard SOCKS port. Although you can use any port number, some programs will only work if you use 1080. -C enables compression, which speeds the tunnel up when proxying mainly text-based information (like web browsing), but can slow it down when proxying binary information (like downloading files).

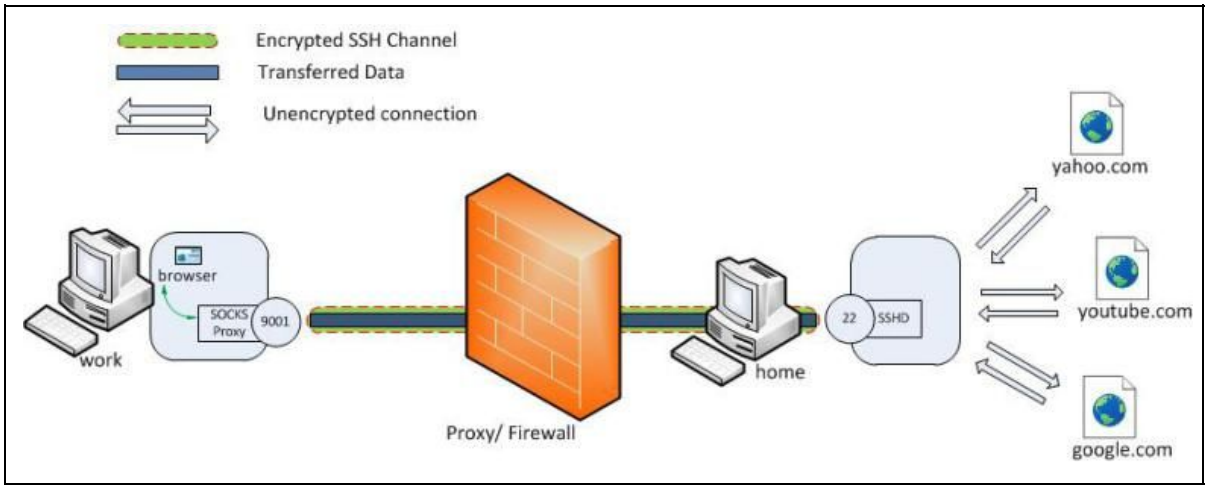
Next you would tell Firefox to use your proxy:

- go to Edit -> Preferences -> Advanced -> Network -> Connection -> Settings...
- check "Manual proxy configuration"
- make sure "Use this proxy server for all protocols" is cleared
- clear "HTTP Proxy", "SSL Proxy", "FTP Proxy", and "Gopher Proxy" fields
- enter "127.0.0.1" for "SOCKS Host"
- enter "1080" (or whatever port you chose) for Port.

You can also set Firefox to use the DNS through that proxy, so even your DNS lookups are secure:

- Type in about:config in the Firefox address bar
- Find the key called "network.proxy.socks_remote_dns" and set it to true

The SOCKS proxy will stop working when you close your SSH session. You will need to change these settings back to normal in order for Firefox to work again.
To make other programs use your SSH proxy server, you will need to configure each program in a similar way.



[work]$ ssh -D 9001 home

SSH VPN

SSH-BASED VIRTUAL PRIVATE NETWORKS

ssh contains support for Virtual Private Network (VPN) tunnelling using the tun(4) network pseudo-device, allowing two networks to be joined securely. The sshd_config(5) configuration option PermitTunnel controls whether the server supports this, and at what level (layer 2 or 3 traffic).

The following example would connect client network 10.0.50.0/24 with remote network 10.0.99.0/24 using a point-to-point connection from 10.1.1.1 to 10.1.1.2, provided that the SSH server running on the gateway to the remote network, at 192.168.1.15, allows it.

On the client:

```
# ssh -f -w 0:1 192.168.1.15 true
# ifconfig tun0 10.1.1.1 10.1.1.2 netmask 255.255.255.252
# route add 10.0.99.0/24 10.1.1.2
```

On the server:

```
# ifconfig tun1 10.1.1.2 10.1.1.1 netmask 255.255.255.252
# route add 10.0.50.0/24 10.1.1.1
```

Client access may be more finely tuned via the /root/.ssh/authorized_keys file (see below) and the PermitRootLogin server option.
The following entry would permit connections on tun(4) device 1 from user “jane” and on tun device 2 from user “john”, if PermitRootLogin is set to “forced-commands-only”:

```
tunnel="1",command="sh /etc/netstart tun1" ssh-rsa ... jane
tunnel="2",command="sh /etc/netstart tun2" ssh-rsa ... john
```

Since an SSH-based setup entails a fair amount of overhead, it may be more suited to temporary setups, such as for wireless VPNs. More permanent VPNs are better provided by tools such as ipsecctl(8) and isakmpd(8).