« Monitoring : Making sense of the buzz word jungle

Learn by Errors : Java + OSGi »

SSH Tunneling Explained

March 21, 2012 by Buddhika Chamith

Recently I wanted to set up a remote desktop sharing session from home pc to my laptop. While going through the set up guide I came across ssh tunneling. Even though there are many articles on the subject still it took me a considerable amount of googling, some experimenting and couple of Wireshark sessions to grasp what's going under the hood. Most of the guides were incomplete in terms of explaining the concept which left me desiring for a good article on the subject with some explanatory illustrations. So I decided to write it my self. So here goes…

Introduction

A SSH tunnel consists of an encrypted tunnel created through a SSH protocol connection. A SSH tunnel can be used to transfer unencrypted traffic over a network through an encrypted channel. For example we can use a ssh tunnel to securely transfer files between a FTP server and a client even though the FTP protocol itself is not encrypted. SSH tunnels also provide a means to bypass firewalls that prohibits or filter certain internet services. For example an organization will block certain sites using their proxy filter. But users may not wish to have their web traffic
monitored or blocked by the organization proxy filter. If users can connect to an external SSH server, they can create a SSH tunnel to forward a given port on their local machine to port 80 on remote web-server via the external SSH server. I will describe this scenario in detail in a little while.

established, the user can connect to earlier specified port at first machine to

JKJ

Search

ARCHIVES

August 2012 (1)
July 2012 (1)
June 2012 (1)
May 2012 (2)
March 2012 (2)
February 2012 (2)

access the network service.

Port Forwarding

SSH tunnels can be created in several ways using different kinds of port forwarding mechanisms. Ports can be forwarded in three ways.

1. Local port forwarding
2. Remote port forwarding
3. Dynamic port forwarding

I didn't explain what port forwarding is. I found Wikipedia's definition more explanatory.

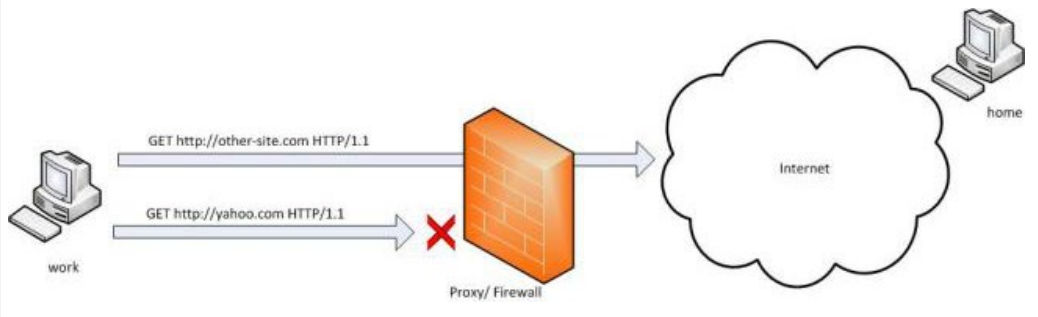
> Port forwarding or port mapping is a name given to the combined technique of
>
> 1. translating the address and/or port number of a packet to a new destination
> 2. possibly accepting such packet(s) in a packet filter(firewall)
> 3. forwarding the packet according to the routing table.

Here the first technique will be used in creating an SSH tunnel. When a client application connects to the local port (local endpoint) of the SSH tunnel and transfer data these data will be forwarded to the remote end by translating the host and port values to that of the remote end of the channel.

So with that let's see how SSH tunnels can be created using forwarded ports with an examples.

Tunnelling with Local port forwarding

Let's say that yahoo.com is being blocked using a proxy filter in the University. (For the sake of this example. :). Cannot think any valid reason why yahoo would be blocked). A SSH tunnel can be used to bypass this restriction. Let's name my machine at the university as 'work' and my home machine as 'home'. 'home' needs to have a public IP for this to work. And I am running a SSH server on my home machine. Following diagram illustrates the scenario.



To create the SSH tunnel execute following from 'work' machine.

```
1 ssh -L 9001:yahoo.com:80 home
```

September 2011 (1)
August 2011 (2)
July 2011 (1)
June 2011 (1)
May 2011 (6)
November 2010 (1)
October 2010 (4)
May 2010 (1)
April 2010 (1)
March 2010 (4)
October 2009 (2)
September 2009 (1)
August 2009 (2)
July 2009 (4)

CATEGORIES

Concepts (10)
FOSS (1)
Guides (1)
Hadoop (2)
How To's (7)
miscellaneous (5)
Projects (4)
rants (3)
Reviews (1)
Security (4)
SOA (4)
Tips (19)
Uncategorized (2)

Apache Cassandra Apache Hadoop Apache Hive Apache ODE Apache Thrift API Asynchronous I/O Axis2 BAM BASH Bindings Bloom filter BND BPEL BPM Builds Business Activity Monitor Cassandra cassandra-cli CEP command line Debut distributed Equinox extra mile Felix Hadoop Hector Introduction Java JMS JMX KPI map-reduce MapReduce Maven nio non blocking NoSQL Operational Intelligence OSGi p2

The 'L' switch indicates that a local port forward is need to be created. The switch syntax is as follows.

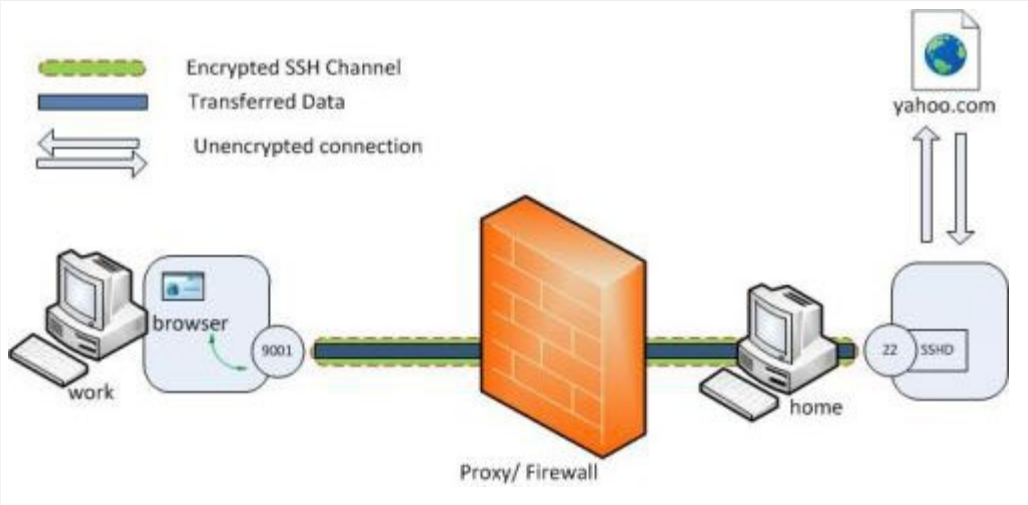
```
-L <local-port-to-listen>:<remote-host>:<remote-port>
```

Now the SSH client at 'work' will connect to SSH server running at 'home' (usually running at port 22) binding port 9001 of 'work' to listen for local requests thus creating a SSH tunnel between 'home' and 'work'. At the 'home' end it will create a connection to 'yahoo.com' at port 80. So 'work' doesn't need to know how to connect to yahoo.com. Only 'home' needs to worry about that. The channel between 'work' and 'home' will be encrypted while the connection between 'home' and 'yahoo.com' will be unencrypted.

Now it is possible to browse yahoo.com by visiting http://localhost:9001 in the web browser at 'work' computer. The 'home' computer will act as a gateway which would accept requests from 'work' machine and fetch data and tunnelling it back. So the syntax of the full command would be as follows.

```
ssh -L <local-port-to-listen>:<remote-host>:<remote-port> <gateway>
```

The image below describes the scenario.



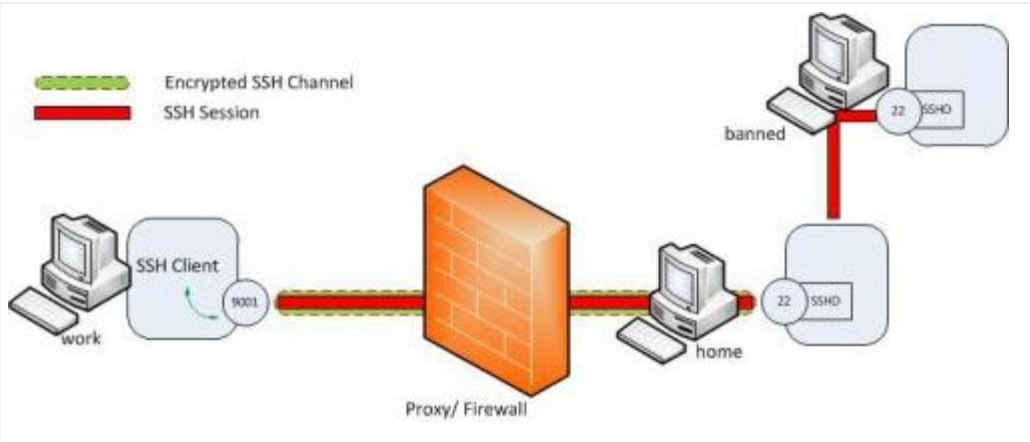
Here the 'host' to 'yahoo.com' connection is only made when browser makes the request not at the tunnel setup time.

It is also possible to specify a port in the 'home' computer itself instead of connecting to an external host. This is useful if I were to set up a VNC session between 'work' and 'home'. Then the command line would be as follows.

```
ssh -L 5900:localhost:5900 home (Executed from 'work')
```

So here what does localhost refer to? Is it the 'work' since the command line is executed from 'work'? Turns out that it is not. As explained earlier is relative to the gateway ('home' in this case) , not the machine from where the tunnel is initiated. So this will make a connection to port 5900 of the 'home' computer where the VNC client would be listening in.

The created tunnel can be used to transfer all kinds of data not limited to web browsing sessions. We can also tunnel SSH sessions from this as well. Let's assume there is another computer ('banned') to which we need to SSH from within University but the SSH access is being blocked. It is possible to tunnel a SSH session to this host using a local port forward. The setup would look like this.



As can be seen now the transferred data between 'work' and 'banned' are encrypted end to end. For this we need to create a local port forward as follows.

```
ssh -L 9001:banned:22 home
```

Now we need to create a SSH session to local port 9001 from where the session will get tunneled to 'banned' via 'home' computer.

```
ssh -p 9001 localhost
```

With that let's move on to next type of SSH tunnelling method, reverse tunnelling.

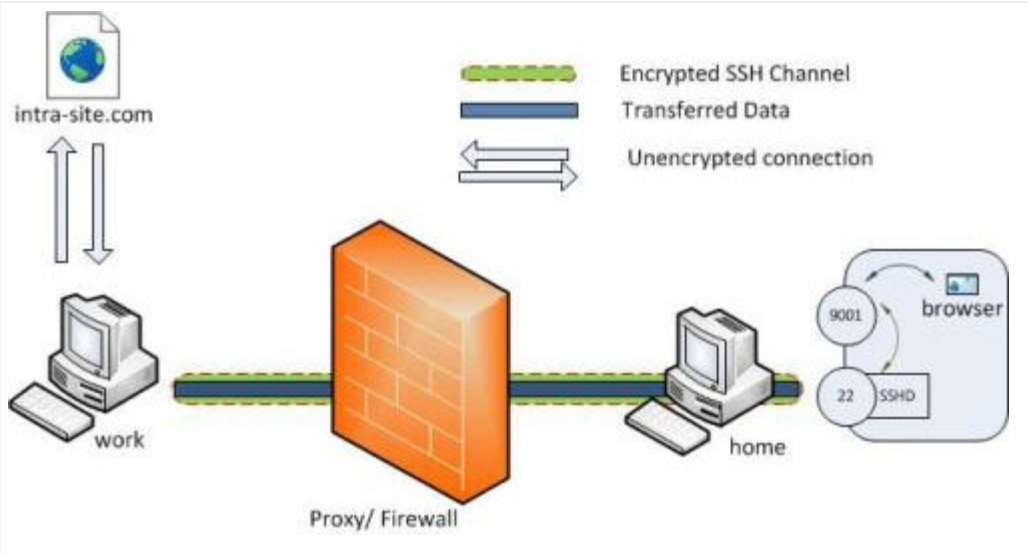
Reverse Tunnelling with remote port forwarding

Let's say it is required to connect to an internal university website from home.
The university firewall is blocking all incoming traffic. How can we connect from 'home' to internal network so that we can browse the internal site? A VPN setup is a good candidate here. However for this example let's assume we don't have this facility. Enter SSH reverse tunnelling..

As in the earlier case we will initiate the tunnel from 'work' computer behind the firewall. This is possible since only incoming traffic is blocking and outgoing traffic is allowed. However instead of the earlier case the client will now be at the 'home' computer. Instead of -L option we now define -R which specifies
a reverse tunnel need to be created.

```
ssh -R 9001:intra-site.com:80 home (Executed from 'work')
```

Once executed the SSH client at 'work' will connect to SSH server running at home creating a SSH channel. Then the server will bind port 9001 on 'home' machine to listen for incoming requests which would subsequently be routed through the created SSH channel between 'home' and 'work'. Now it's possible to browse the internal site
by visiting http://localhost:9001 in 'home' web browser. The 'work' will then create a connection to intra-site and relay back the response to 'home' via the created SSH channel.



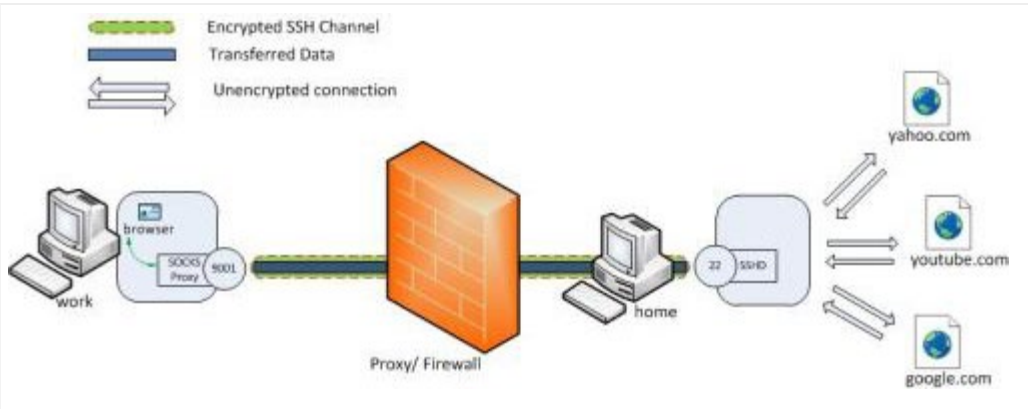
As nice all of these would be still you need to create another tunnel if you need to connect to another site in both cases. Wouldn't it be nice if it is possible to proxy traffic to any site using the SSH channel created? That's what dynamic port forwarding is all about.

Dynamic Port Forwarding

Dynamic port forwarding allows to configure one local port for tunnelling data to all remote destinations. However to utilize this the client application connecting to local port should send their traffic using the SOCKS protocol. At the client side of the tunnel a SOCKS proxy would be created and the application (eg. browser) uses the SOCKS protocol to specify where the traffic should be sent when it leaves the other end of the ssh tunnel.

```
1 ssh -D 9001 home (Executed from 'work')
```

Here SSH will create a SOCKS proxy listening in for connections at local port 9001 and upon receiving a request would route the traffic via SSH channel created between 'work' and 'home'. For this it is required to configure the browser to point to the SOCKS proxy at port 9001 at localhost.



Share this: