

Design of jttt - "Java TCP Tunneling Tool" version 2.0

Date: 08/25/2003

Developers (user names in sourceforge.net)

Jeffkorn

jiechau

Outline:

1. Introduction
2. Related work
3. The Project Design of jttt 2.0
4. The Implementation
5. Bugs and Further Work.

Appendix

- A1. How to configure "Public Key Authentication" in SSH Server and Clients
- A2. Common setup for SSH Server to allow Port Forwarding
- A3. Common setup for SSH Client Tools to do the Port Forwarding.
- A4. How to use VNC software (RealVNC)

Reference

1. Introduction

TCP protocol [RFC 793] is widely used in the current internet. Most services we used in our daily life are on top of TCP protocol, e.g. the HTTP protocol for browsing www pages, the FTP protocol for file transfer, and RFB protocol ("remote framebuffer") for virtual network computing (used in so called "remote desktop software"). Some big issues about all these TCP services are: their data transferred in the wide internet is not secure (bytes stream, or actually, plain text data in bytes stream). And, when the local user network environment is inside a Firewall, some services can't do the firewall

penetration. These issues all limit the usage and convenience for TCP services.

Our project is to construct a network mechanism to enable the secure data transfer of these TCP service, and extend the usage of these TCP services between Firewalls.

We will use VNC software as our test TCP service. VNC software (www.realvnc.com) software is a remote control software which allows users to view and interact with one computer (the "Server") using a simple program (the "Viewer") on another computer anywhere on the Internet.

2. Related works

Here is some basic domain knowledge/environment regarding our project:

(1). TCP/IP and Firewalling

Most functionalities about “Internet” we used in our daily life are those TCP services (on top of TCP Protocol, which is on top of IP Protocol), e.g. HTTP Protocol, TELNET Protocol, and SSH Protocol. The common feature of these “client-server” services is that the Server will listen on a specific TCP port, waiting for the Client to connect to. Once the connection is built, data is transmitted between a single TCP port (a socket) in Client and a single TCP port (the port which is listening) in Server. Based on the “Layer Structure” of TCP/IP protocol, the applications can communicate with each other no matter how you redirect/forwarding the transmitted data in the midway, only if you make sure the data finally goes to the right socket in the Server.

The Firewall functionalities provided by our LAN devices/administrators vary from environment to environment. “Generally speaking” our LAN router would inspect the Layer 3 IP header, and sometimes Layer 4 TCP (or UDP) header to determine whether a given packet is forwarded or dropped, which is called the Packet Filtering [Kurose 2002]. Thus it by default drops all outside-in traffic but allows certain kinds of inside-out traffic. Data transmission through SSH protocol (default TCP port 22) is always one of these certain kinds of allowed inside-out connection.

(2).Port Forwarding

In addition to general TCP “Port Mapping” [Douglas 1996] (ports mapping between 2 machines), the “Port Forwarding” technique uses an Intermediate Machine to achieve the Port Mapping functionality. All data which goes to a socket in a machine will be forwarded to another socket in another machine, through this Intermediate Machine. In the normal case, Port Forwarding always used to forward incoming data in a public machine to a local machine (which is inside the firewall) for those typical “client-server” TCP services. This way you can build a service in a local machine but listen on a public machine, as if the service is built on the public machine. For more information you could reference the following chapter ((3).SSH - “Secure Shell” – “(v).About the Port Forwarding functionality provided by SSH”).

(3).SSH - “Secure Shell”

(i). About SSH:

SSH -“Secure Shell” is a program to login or to execute commands in another computer. It is intended as a replacement for telnet and those r-commands (e.g. rlogin, rsh, and rcp) because SSH provides strong authentication and secure communications over unsecure channels. SSH is a TCP service, which means it is a typical “client-server” TCP service (the SSH server default listens on port 22 for connection). There are two versions of Secure Shell available: SSH1 and SSH2. We will only use SSH2 in our project.

(ii).About the authentication methods:

SSH2 provides 3 authentication methods (not like TELNET protocol, there is only “password authentication”). These methods are: Password authentication, Public Key authentication, and Keyboard-interactive authentication. In this project, we will only use the Public Key authentication method.

(iii).About the Protocol:

SSH uses the “Secure Shell Protocol”. The SSH Communications Security (<http://www.ssh.com>) is the developer of Secure Shell (secsh) Protocol and maintains the official releases of SSH1 and SSH2.

(iv).About the Tools:

The “SSH Secure Shell” (SECSH) is the SSH Communication Security's official

release (implementation) of Secure Shell. Another famous unofficial implementations of SSH is OpenSSH (<http://www.openssh.org>). Both provide Server side and Client side tools. There are still some famous SSH Client software, like PuTTY, which is used for SSH Client to build connection to the SSH Server.

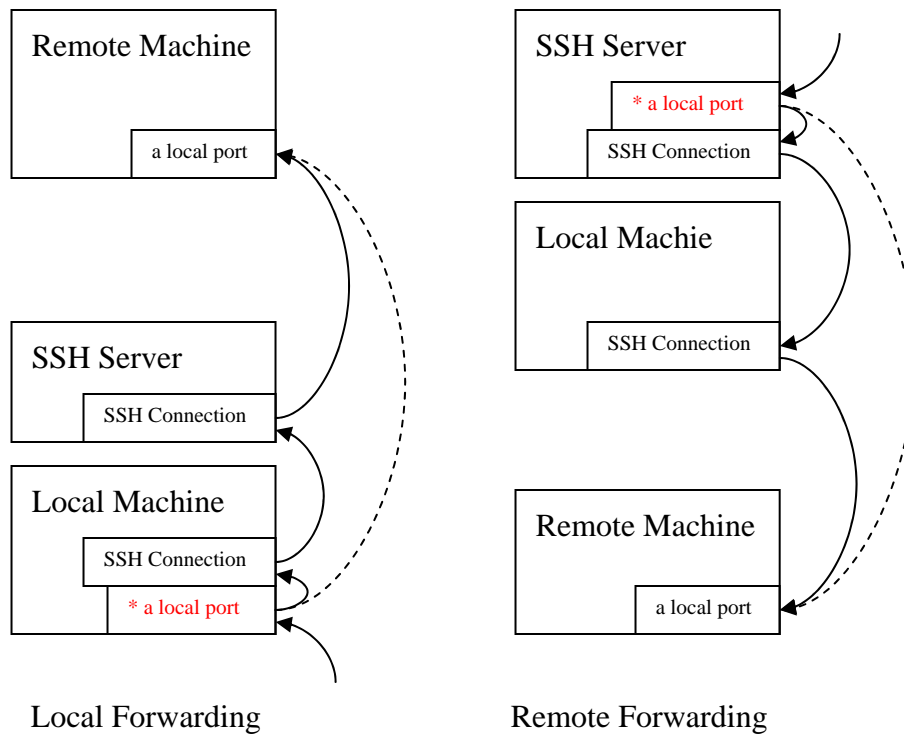
(v).About the Port Forwarding functionality provided by SSH:

SSH has Port Forwarding functionality. It simply encrypts your forwarding data through the SSH connection. Two terms associated to Port Forwarding provided by SSH. They are “Local Forwarding” and “Remote Forwarding”.

Local Forwarding: The **SSH Client** will allocate a local port (via your specification) in your Local Machine to listen on. Data which goes into this specific local port will be encrypted, passed through the SSH connection to the SSH server, then the SSH server decrypts these data and then forwards it to a desired remote port in a Remote Machine.

Remote Forwarding: The **SSH Server** will allocate a local port (via your specification) in the SSH Server machine to listen on. Data which goes into this specific port in the SSH server machine will be encrypted, passed through SSH connection back to the Local Machine (the SSH Client), then the local SSH Client decrypts the data and forwards it to a desired port in a Remote Machine.

For the implementations, you could reference Appendix: “A3. Common setup for SSH Client Tools to do the Port Forwarding.” about how to assign Port Forwarding using SSH Clients.



* The Ports in RED color is allocated by the SSH Client or Server

(vi).More information, please reference:

<http://www.onsight.com/faq/ssh/ssh-faq.html>

<http://www.ssh.com>

<http://www.openssh.com>

<http://www.ietf.org/ids.by.wg/secsh.html>

<http://www.chiark.greenend.org.uk/~sgtatham/putty>

(4).JAVA2 SSH Tools API

SSHTools is a suite of Java applications providing both SSH client and server functionality as well as an extensible development library. JAVA programmer can include this library to performance SSH functions in the JAVA program. This library provides SSH functionality including SSH connection, SSH authentication (password, public key, and Keyboard-interactive), and SSH Port Forwarding.

More information, please reference:

<http://www.sshtools.com/>

<http://sourceforge.net/projects/sshtools>

(5).VNC

VNC stands for Virtual Network Computing. RealVNC (www.realvnc.com) is the official home of VNC, staffed by the original team who created and developed it whilst at AT&T. The VNC system is based on the concept of a remote framebuffer or RFB. RealVNC is a remote control software which allows you to view and interact with one computer (the "server") using a simple program (the "viewer") on another computer (Please reference Index: "A3. How to use VNC software (RealVNC)").

VNC is a TCP service, which means the data transmission between VNC Client and VNC Server are on top of TCP/IP protocol.

More information, please reference:

<http://www.realvnc.com>

http://vega.lpl.arizona.edu/vnc_docs/protocol.html

http://vega.lpl.arizona.edu/vnc_docs/rfbproto.pdf

3. The Project Design of jttt 2.0

In a nutshell, we want to do the Port Forwarding between 2 machines (our Client Machine and the Server Machine). While these 2 machines are resided inside their own Firewall, we need the 3rd Machine (the Intermediate Machine) as the intermediate. The Intermediate machine provides SSH authorization and SSH Port Forwarding functionality. You only need to have a user account in the Intermediate Machine then you could connect to the Intermediate Machine from Client Machine or Server Machine, and achieve the Port Forwarding Functionality at the same time when you build SSH connection.

You have only one user account in the Intermediate Machine, but the required Port Forwarding matches might be many (e.g. when you are required to forward

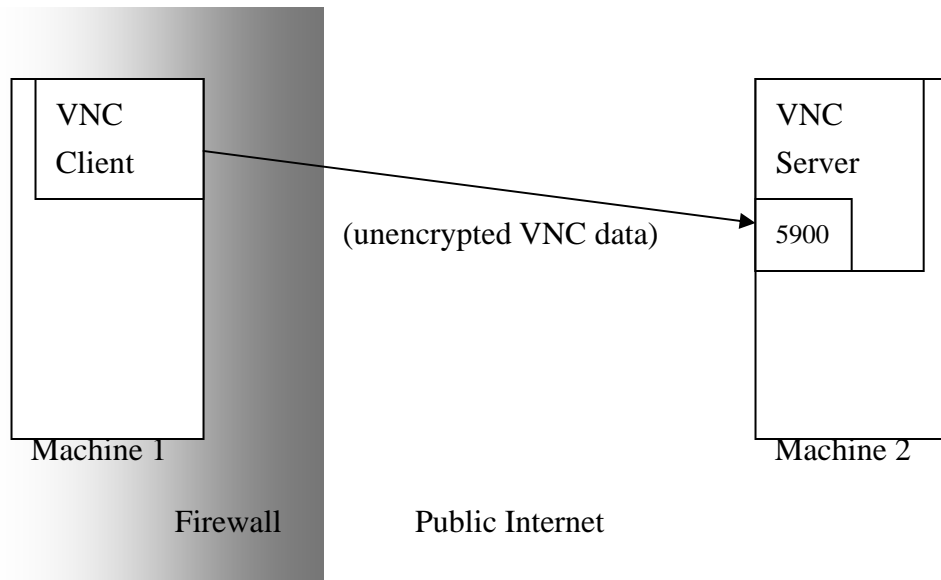
Server_1:Port_1 to Client_1:Port_1, and Server_2:Port_2 to Client_2:Port_2, and so on and so forth). You could achieve all these Port Forwarding jobs via that same UNIX account in the Intermediate Machine. We use an authorization method named “jttt username/password” to distinguish different Server-Client Port Forwarding pairs. This “jttt username/password” is maintained in the Intermediate Machine and used for Client and Server Machines to do the authorization, retrieve the Private Keys, and use the retrieving Private Key to distinguish different Server-Client Port Forwarding pairs.

Here I will use the VNC software as a TCP service example to demonstrate the design of our project, jttt – Java TCP Tunneling Tool version 2.0. The VNC software is used to achieve the remote desktop control. The VNC Client will try to connect to the VNC Server Port 5900, and all data is transferred via this TCP socket connection.

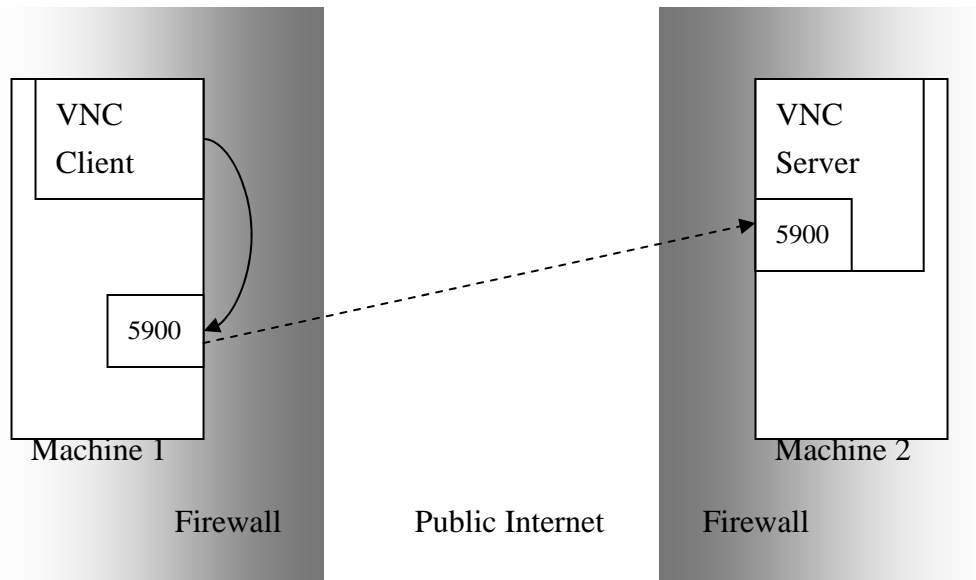
(1).The Design: modify the VNC software setting

Our goal is to design a TCP port forwarding mechanism and apply it on the VNC software (a typical client-server TCP service, so the same method can applied on other TCP services, e.g. HTTP, TELNET, ... etc.). The basic idea is to modify the VNC software setting to let it match our TCP Port Forwarding mechanism. First we talk about how to modify the VNC software setting.

Normal VNC connection (as well as any other TCP service connection) is built when both Client and Server is exposed in the public Internet. Sometimes even the Client resides inside a firewall (Under the “common” firewall environment as mentioned in previous chapter) the TCP connection can be built without any limitation. Data without any encryption is transmitted through a direct connection. This situation is illustrated as following diagram:



But when the Server is resided inside a firewall, we need an Intermediate Machine to be the gateway between VNC Client and VNC Server. Regardless the Intermediate Machine, we now let the VNC Client connect to its local port 5900 (instead of connecting to remote VNC Server) as if there is a pseudo VNC Server installed on its own machine listening on local port 5900. Then the remaining problem is “how to transmit data from VNC Client port 5900 to the VNC Server port 5900, when both Client and Server is inside firewalls”.



The method about how to transmit data from VNC Client port 5900 to the VNC Server port 5900 is mentioned below.

(2).The Design: TCP Port Forwarding mechanism

We use the SSH connections, letting both VNC Client and VNC Server connect to an Intermediate Machine, to achieve the TCP Port Forwarding, firewall penetration, and encryption of transmitted data.

The idea is, the Intermediate Machine is running SSH server, listening on port 22, and both VNC Client and VNC Server uses the Web browser (e.g. Microsoft IE) with JAVA plug-in to run the Java Applet, and the Java Applet program is meant to be the SSH client, to be used to build SSH connection to the Intermediate Machine (and achieve the Port Forwarding at the same time).

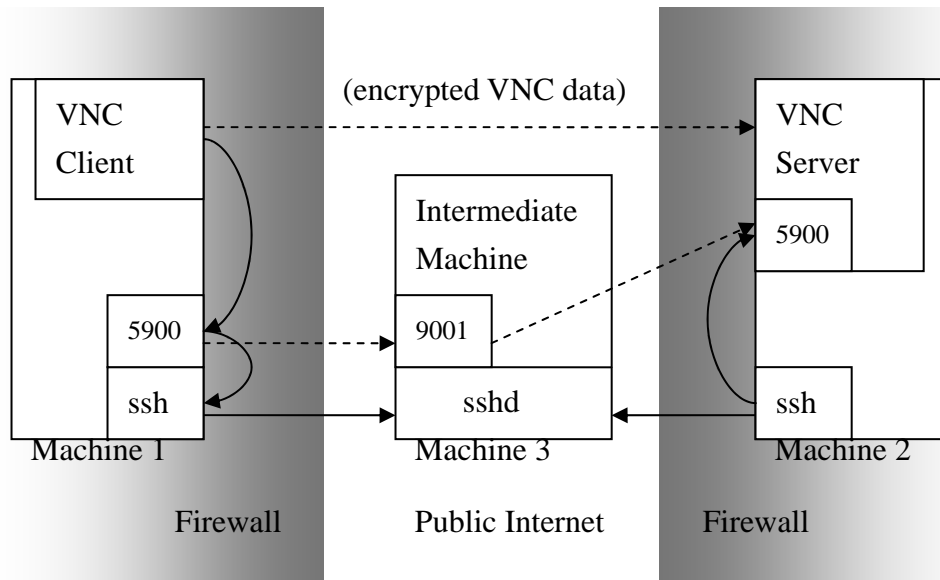
We first build SSH connection from VNC Server to the Intermediate Machine. Via the Port Forwarding functionality (the “Remote Forwarding”) provided by SSH connection, we could allocate a Port in SSH Server (the Intermediate Machine), say, port 9001, to listen on. And assign the forwarding target to the VNC Server port 5900. Any connection to the port 9001 in Intermediate Machine would be forwarded to

VNC Server port 5900. Notice that if you reference the diagram in previous chapter ((3).SSH - “Secure Shell” – “(v).About the Port Forwarding functionality provided by SSH”), currently you should look at the “Remote Forwarding” diagram. The Intermediate Machine is the SSH Server in the diagram. The VNC Server is the Client Machine in the diagram. And the VNC Server happens to be the Remote Machine in the diagram.

Then we build SSH connection from VNC Client to the Intermediate Machine. Via the port forwarding functionality (the “Local Forwarding”) provided by SSH connection, we could allocate a port 5900 on the local machine (the VNC Client) to listen on. And assign the forwarding target to the Intermediate Machine port 9001. Any connection to the VNC Client port 5900 would be forwarded to port 9001 on the Intermediate Machine. Notice that if you reference the diagram in previous chapter ((3).SSH - “Secure Shell” – “(v).About the Port Forwarding functionality provided by SSH”), currently you should look at the “Local Forwarding” diagram. The Intermediate Machine is the SSH Server in the diagram. The VNC Client is the Client Machine in the diagram. And the Intermediate Machine happens to be the Remote Machine in the diagram.

Under this scenario, both VNC Client and VNC Server can be resided inside firewalls, and the forwarding data is encrypted. Once these tunnels are built, the application level connection from VNC Client to its local port 5900 (mentioned in previous “The Design: modify the VNC software setting”) is actually a connection to VNC Server Port 5900.

Notice that under this scenario the Intermediate Machine should reside in a public Internet address (or at least where both VNC Client and Server can connect to). The whole scenario is illustrated as follow:



(3).The Design: Management of user session

When we build SSH connection from VNC Client (or VNC Server) to the Intermediate Machine, we use the “Public Key authentication method” (associated to a valid user account in Intermediate Machine) provided by SSH protocol. If there are lots of requests (e.g. many different VNC Client/Server connection pairs through a same Intermediate Machine), we can’t just create a new user account in the Intermediate Machine for each connection pair. Thus we need a mechanism to manage those connections.

Here we use the “Forced Command” feature in SSH protocol to solve it. We still use only one user account in the Intermediate Machine to handle all the SSH connection requests, but give different public key pair to different VNC Client/Server pair, with different public key associated to different “Forced Command”. The Forced command is actually to run a small Shell script in the Intermediate Machine, named “keepAlive.sh” with arguments indicated the different VNC Client/Server pair and different port for forwarding (start from port 9001 for each pair). The remaining problem is how to deliver the “authenticated key” to VNC Client and VNC Server.

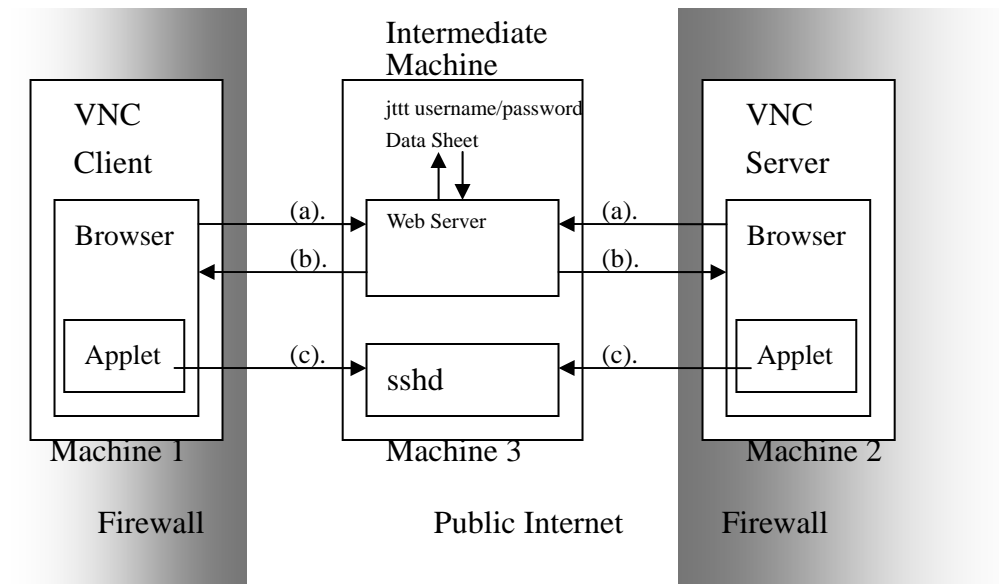
Since there is only one account used for SSH connection in the Intermediate Machine,

we need a username/password data sheet to distinguish different Client/Server pair. We call it “jttt username/password”. For every user who has the access to use this port forwarding mechanism, we assign a specific jttt username/password to him/her. And in the Intermediate Machine we run a program to do the authentication of “jttt username/password” job (it is the “Intermediate.java” program in our implementation, see next chapter) and deliver the “Private Key” to user when authenticated.

(4).A Web server in the Intermediate Machine

The one less thing we need is a Web server in the Intermediate Machine to do the “jttt username/password” authentication work and deliver the Applet Program the Client or Server Machines.

When a user wants to build the connection, he first uses the “jttt username/password” to access the Intermediate Machine Web pages. After authenticated, he can run the Java Applet program in his local browser to achieve the SSH connection and Port Forwarding function. The Intermediate Machine will assign a Private Key to user and register associated data (e.g. Forced Command, the Forward Port used) for that key. User retrieves that key and then uses that key to do the Public Key Authentication Method to build SSH connection to the Intermediate and the port forwarding function via Java Applet program.



- (a). Use jttt username/password to grant access
- (b). Retrieve Private Key
- (c). Build SSH connection (using retrieved Key) and Port Forwardings

4. The Implementation

(1).Software/Tools used.

The VNC software:

The software to demonstrate the TCP service, installed in server and client.

<http://www.realvnc.com>

SECSH (SSH Secure Shell) or OpenSSH:

The SSH server is installed in the Intermediate machine, for both client and server to connect to. In the jttt 2.0 release, we plan to use SECSH only. The further release will support OpenSSH.

<http://www.openssh.org>

Java 2 Platform, Standard Edition (J2SE) version 1.4, SDK:

Used to write the JAVA Applet programs.

<http://java.sun.com>

Java2 SSH Tools:

The SSH client tool library (API) in JAVA programming language. This library is used to perform the SSH Client functionality in both VNC Client and Server side.

<http://sourceforge.net/projects/sshtools>

Korn Shell 93:

Used to write the shell scripts and web CGI scripts in the Intermediate machine.

<http://www.cs.princeton.edu/~jlk/kornshell/doc/man93.html>

Perl v5.8.0:

Used to write the web CGI scripts in the Intermediate machine (as an alternative choice other than Korn Shell CGI scripts).

<http://www.perl.com>, <http://www.perldoc.com>

Apache Ant 1.5.3:

Used to manage the project sources and install jttt 2.0.

<http://ant.apache.org>

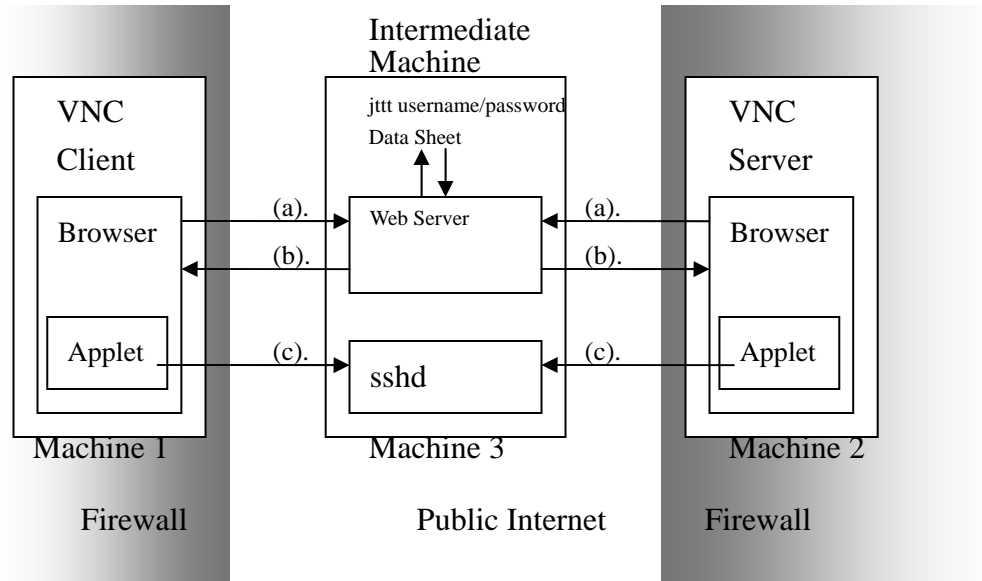
NetBeans™ IDE 3.5:

Used for the development of Java source codes..

<http://www.netbeans.org>

(2). The Implementation

The user authentication is achieved via web pages in the Intermediate Machine and the Port Forward mechanism is achieved via running the associated JAVA Applet programs in the Client of Server Machines (and there is a SSH Server in the Intermediate Machine). In the example of VNC software, the relation is illustrated as follow:



- (a). Use jttt username/password to grant access
- (b). Retrieve Private Key
- (c). Build SSH connection (using retrieved Key) and Port Forwardings

Intermediate Machine runs:

- (i). The SSH server
- (ii). A JAVA program Intermediate.java. (for assign keys and used ports)
- (iii). A Web Server

Server and Client Machines:

- (i). Only needs the web browser (with Java Plug-in, version 1.4 or above)
- (ii). VNC software (our test TCP service).

For example, we use sparky.cs.nyu.edu as the Intermediate Machine. It runs a Web Server, a SSH server, and the program Intermediate.java to listen on port:9000.

The jttt user (either from VNC Server or Client machines) first uses web browser to login the Intermediate Machine (web pages, use jttt username/password). After the authentication, he/she could assign the port (in his local machine) to be forward (“Remote” or “Local” forwarding) via the web page. The CGI script in Intermediate Machine (tcpclient.pl or tcpclient.sh) handles the request, tries to connect to localhost jttt port (in the example, port:9000, it’s the “sInterMediaServerPort” setting in *build.properties* file, in the *UserDoc of jttt 2.0*). The Intermediate.java (listening on

port:9000) will deliver the private key back to tcpclient.pl CGI script and assigns a port to be used for forwarding in Intermediate Machine (these ports start from 9001). If everything goes well, the CGI script will redirect the user web browser to execute a Java Applet program, which contains the private key and associated forwarding information. The user executes the Applet program to connect Intermediate Machine SSH Server and achieve the Port Forwarding function.

When the user is running this Applet program successfully, he could build the SSH connection to Intermediate Machine. This SSH connection will invoke a 'Forced Command' (instead of a shell) in the Intermediate Machine, and further invoke the keepAlive.sh in the Intermediate Machine to update a data file mentioning the state of this SSH connection and port forwarding data (so when the Intermediate Machine is doing the 'check user session' work, the ssh client connection won't be cleaned).

The Intermediate.java has a Thread named "CheckSession" (an inner class: Intermediate\$CheckSession.class), which is doing the works of checking the quantity of prepared Key pairs, and checking the user sessions (if alive). It will check the data file modified by keepAlive.sh to see if a user session is still available.

(3). The VNC Software example

In our example, we use a desktop PC running on Microsoft Windows XP as the VNC Server (the Server Machine), a desktop PC running on Linux RedHat 9 as the VNC Client (the Client Machine). Both of them are reside inside their own firewalls. The Intermediate Machine in our example is a Sun Sparc Machine running on Solaris 8, with public hostname: sparky.cs.nyu.edu.

The VNC Server first launch its service (the VNC Server software) listening on its local port 5900, then uses a browser pointing to the URL of Intermediate Machine to open the web page, does the authentication (jttt username/password), and assigns the local port (5900) to be forwarded (the "Remote Forwarding"). The Intermediate then allocate a local port (say, port:9001) in Intermediate to forward the connection to VNC Server. The VNC Server then executes the Java Applet with all the above information and setting.

The VNC Client can now uses a browser pointing to the URL of Intermediate

Machine to open the web page, does the authentication (jttt username/password), choose the service (which Server Machine to connect to) from the list in web pages, and assigns the local port (say, 5900) to be forwarded (the “Local Forwarding”). The Intermediate collects all the information/settings, does the necessary setting and redirect the browser to execute the Java Applet with all the information/settings above. When the VNC Client is running the Applet, it will allocate a local port 5900 on its own local machine to listen to. Then user in the Client Machine could run the VNC Client Software (the viewer), connecting to its localhost display number “0” (it means port 5900).

For more information about how to install jttt 2.0 and how to run it, please reference the “*UserDoc of jttt 2.0*”

5. Bugs and Further Work.

(1).Bugs:

Actually this is not a bug but something you should be aware of. The jttt 2.0 uses a jttt port (the “sInterMediaServerPort” setting in build.properties file) to provide the private key content to a Server or Client Machine. This feature is inherited from jttt 1.0. But currently the socket connection through “sInterMediaServerPort” would be only from the localhost. If you are worried about the security issue, you could limit the connection to “only from localhost”. And the methods to do this would be via a OS setup, or a network packages filter software (e.g. “iptables”/”ipchains” in Linux, or “ipfilter” in the BSDs, or software like “tcp wrapper”).

(2).Idea for further work:

In the further work, we wish to fix/modify the bugs mentioned above. The methods could be: modify the jttt program in Intermediate to limit the connect to jttt port from only “from localhost” (the Intermediate.java). Another method is to modify the design of jttt, to use another method to deliver the Private Keys instead of using the TCP connection to deliver the Private Key.

Appendix

A1. How to configure “Public Key Authentication” in SSH Server and Clients

You should create a pair a authentication key first, keep the “Private Key” in your SSH Client, and the “Public Key” in your SSH Server. Then configure your SSH Client and Server setting to recognize and use this key when you do the authentication.

(1).Prepare your authentication keys first

In the following example, I’ll use OpenSSH, SSH Secure Shell and PuTTY as ssh key generation tools. After the key generation procedure you will have a Public Key and Private Key. I’ll use the file “mykey” to store the Private Key and “mykey.pub” to store the Pubic Key in all these 3 example key generation tools.

(i).use OpenSSH

You could use the commands as follows to generate a 1024 bits key pair using RSA or DSA algorithm. The generated private key is named “mykey”, and the public key is “mykey.pub”, all under the current directory:

```
% ssh-keygen -t dsa -b 1024 -f mykey
```

```
% ssh-keygen -t rsa -b 1024 -f mykey
```

(ii). use SSH Secure Shell

You could use the commands as follows to generate a 1024 bits key par using RSA or DSA algorithm. The generated private key is named “mykey”, and the public key is “mykey.pub”, all under the current directory:

```
% ssh-keygen2 -t dsa -b 1024 mykey
```

```
% ssh-keygen2 -t rsa -b 1024 mykey
```

(iii). use PuTTY

Run the GUI application named "puttygen.exe" (should be in the same directory as putty.exe). There you could assign the key algorithm, numbers of bits of the key, and simply click "Generate" to do that. Remember to move your mouse all the time when it is generating the key.

(2).Configure SSH Server to allow "Public Key Authentication"

(i). OpenSSH Server

Server setting:

In the SSH daemon configuration file (default in /etc/ssh/sshd_config) you could add this line to allow Public Key Authentication in SSH2. If you don't specify this value, it is "yes" (to allow Public Key Authentication) by default:

```
PubkeyAuthentication yes
```

User account setting:

Then you should set up your user account to allow Public Key authentication. This is done by copying the public key content (in our example, it's the content of "mykey.pub") to the file \$HOME/.ssh/authorized_keys. In this file, every line is a public key, for example, here is a sample content:

```
ssh-dss AAAAB3Nxxx...
```

Where "ssh-dss", or "ssh-rsa" denotes key types of DSA or RSA, the following AAAAB3Nxxx is the content of public key file (in our example, the whole content of mykey.pub file, except the "#" command lines). If we further apply "Forced Command" function on this key, the format would be:

```
command="echo hi" ssh-dss AAAAB3Nxxx...
```

(ii). SSH Secure Shell Server

Server setting:

In the SSH daemon configuration file (default in /etc/ssh2/sshd_config) you

could add “publickey“ option to the “AllowedAuthentications” field. This is to allow Public Key Authentication in SSH2. If you don’t specify this value, by default it is “yes” (to allow Public Key Authentication):

```
AllowedAuthentications  hostbased,publickey,password
```

User account setting:

Then you should set up your user account to allow Public Key authentication. This is done by keeping the public key (in our example, it’s “mykey.pub”) under the directory \$HOME/.ssh2/. For example, if the user account is “jch280” and the default home directory of this account is /home/jch280 then the public key should be in /home/jch280/.ssh2/mykey.pub.

Then you should edit the file “authorization” (\$HOME/.ssh2/authorization) to let the sshd recognize the public key you specified. This is done by adding these 2 lines into “authorization” file:

```
Key      mykey.pub
Command “echo hi”
```

where “mykey.pub” could be an absolute path of the public key file, and the line “Command” is called Forced Command in SSH Server. This means that when user use this pair of key to authenticate, instead of having a shell to use, he can only execute the command issued in the “Command” line (like above example, it’s only a echo message then logout). If this “Command” line is omitted, by default he will have a shell to use.

(3).Configure SSH Client to use “Public Key Authentication”

Here we introduce how to use SSH Clients to connect to a SSH Server using Public Key authentication:

(i). OpenSSH SSH Client

If you use OpenSSH Client (the tool: ssh) and you store the Private Key in this location /home/jlk/data/mykey, you could use this command to connect to

SSH server: sparky.cs.nyu.edu, using the user account “jch280” and use the key file “mykey”:

```
% ssh -2 -l jch280 -i /home/jlk/data/mykey sparky.cs.nyu.edu -v
```

The extra option “-2” denotes to use SSH2 protocol to connect, “-v” is verbose mode which is useful to trace error.

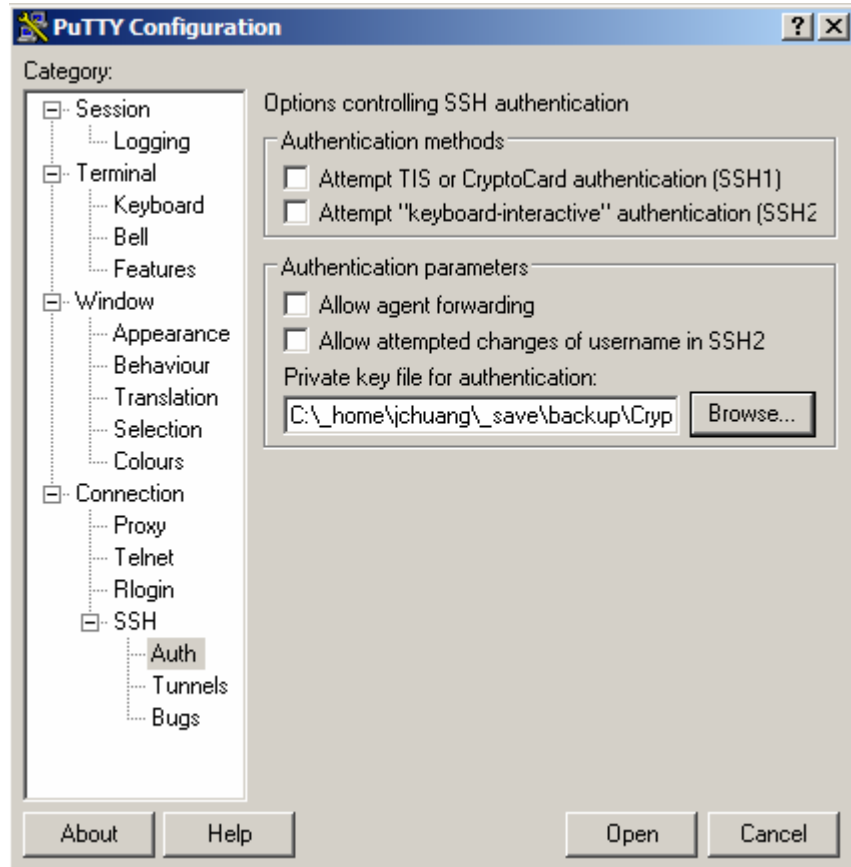
(ii). SSH Secure Shell SSH Client

If you use SSH Secure Shell, the client command is the same as OpenSSH except it doesn't accept the -2 argument:

```
% ssh2 -l jch280 -i /home/jlk/data/mykey sparky.cs.nyu.edu -v
```

(ii). PuTTY SSH Client

You could do this by starting create a “Session Profile” in PuTTY first, let's call it “vnc client 0”. Then Click on the “Auth” (under “Connection” -> “SSH” in the left hand side menu). By checking “Browse...” you could give it a Private Key file. Notice that the PuTTY SSH Client only accept the Private Key file in its own format (which is in .PPK file extension). If you use a private key generated somewhere else (by some other key generation tool other than puttygen.exe) you should use the “puttygen.exe” application to convert to key to the PuTTY format. Finally, remember to go back to “Session” to save this setting associated to the “vnc client 0” session profile.



A2. Common setup for SSH Server to allow Port Forwarding

By default, the SSH Server (both OpenSSH and SECSH) is set to allow you do the Port Forwarding. Also by default the ports, allocated by your Port Forwarding commands, is set to allow connections from localhost only. This part is for the security requirement, and by default you don't need to worry about (the default setting of SSH server would cover this part, unless you have changed the default settings of your SSH server or its behaviors are not what you expected).

(1). You could check the configuration file to make sure the SSH server is set to allow Port Forwarding:

In OpenSSH:

Specify " AllowTcpForwarding" in sshd_config to "yes".

(Reference: http://www.openbsd.org/cgi-bin/man.cgi?query=sshd_config&sektion

=5&arch=&apropos=0&manpath=OpenBSD+Current)

In SECSH:

Specify "AllowTcpForwarding" in sshd_config to "yes".

(Reference:http://www.ssh.com/documents/32/sshd2_configman.txt)

Check:

You could use the following command (in any SSH Client machine) to try to connect to a SSH Server named "Bob":

```
ssh -R 9001:Alice:5900 Bob
```

This will result in allocating a socket to listen to port 9001 on the remote machine Bob, and whenever a connection is made to Bob:9001, the connection is forwarded to Alice:5900. You could logon to Bob and try to telnet to localhost:9001 see if you can connect to the Service on Alice:5900.

(2).Restrict the connection (to opened ports) to only "from localhost".

The default behavior for Port Forwarding is to only bind to localhost (so you can't telnet in from outside) unless you specify a -g on the ssh command line (so don't do it). You could also check the sshd_config configuration file to make sure the SSH server is correctly set (or change this default setting).

In OpenSSH:

Specify "GatewayPorts" in sshd_config to "no".

(Reference:http://www.openbsd.org/cgi-bin/man.cgi?query=sshd_config&sektion=5&arch=&apropos=0&manpath=OpenBSD+Current)

In SECSH:

Specify "ForwardACL" in sshd2_config

(Reference:http://www.ssh.com/documents/32/sshd2_configman.txt)

Check:

In the above "Alice and Bob" example, you could try to connect to Bob:9001 both from localhost (Bob) or from anywhere outside Bob. All the connections from

outside Bob should be rejected, and only those connections from localhost (Bob) should be allowed.

A3. Common setup for SSH Client Tools to do the Port Forwarding.

Here we introduce 3 common SSH Client Tools to achieve SSH Port Forwarding function. They are “OpenSSH”, “SSH Secure Shell” and “PuTTY”.

(1).Use “OpenSSH” or “SSH Secure Shell” SSH Client (under UNIX operating system)

For the “Local Forwarding”, suppose you wish to connection to a machine: “sparky.cs.nyu.edu” with the user account “jch280”, and do the Port Forwarding as: Forward all data which goes into local port 5900 (in you local machine) to port 9001 in sparky.cs.nyu.edu. You can use the follow command:

In OpenSSH:

```
% ssh -L 5900:localhost:9001 -l jch280 sparky.cs.nyu.edu
```

In SSH Secure Shell:

```
% ssh2 -L 5900:localhost:9001 jch280@sparky.cs.nyu.edu
```

For the “Remote Forwarding”, suppose you wish to connection to a machine: “sparky.cs.nyu.edu” with the user account “jch280”, and do the Port Forwarding as: Forward all data which goes into port 9001 in sparky to you local port 5900 in you local machine. You can use the follow command:

In OpenSSH:

```
% ssh -R 5900:localhost:9001 -l jch280 sparky.cs.nyu.edu
```

In SSH Secure Shell:

```
% ssh2 -R 5900:localhost:9001 jch280@sparky.cs.nyu.edu
```

For more information, please reference:

<http://www.openbsd.org/cgi-bin/man.cgi?query=ssh>

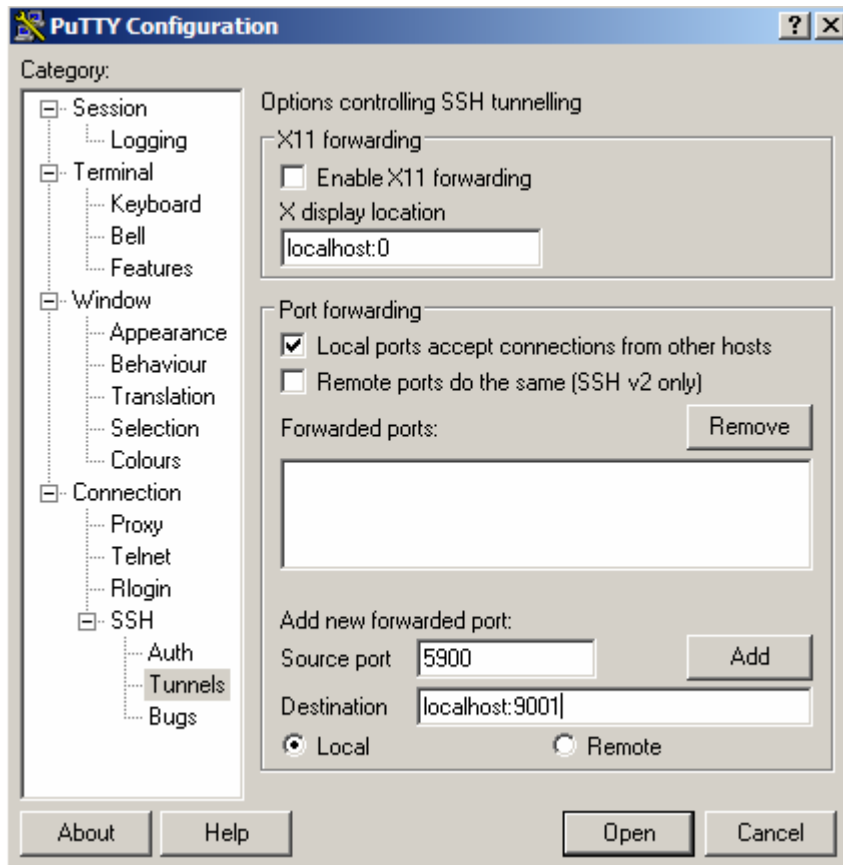
http://www.ssh.com/support/documentation/online/ssh/adminguide/24/Port_Forw

arding.html

(2).Use PuTTY SSH Client Tool (under Microsoft Windows operation system)

For the “Local Forwarding”:

Suppose you wish to connection to a machine: “sparky.cs.nyu.edu” with the user account “jch280”, and do the Port Forwarding as: Forward all data which goes into local port 5900 in you local machine to port 9001 in sparky.cs.nyu.edu. You could do this by starting create a “Session Profile” in PuTTY, let’s call it ”vnc client 0”. Then Click on the “Tunnels” (under “Connection” -> “SSH”) in the left hand side menu. Check “Local ports accept connection from other hosts” and input the associated Port Forwarding data. Remember to check the “Local” in the bottom to indicate that this is a “Local Forwarding” setting:

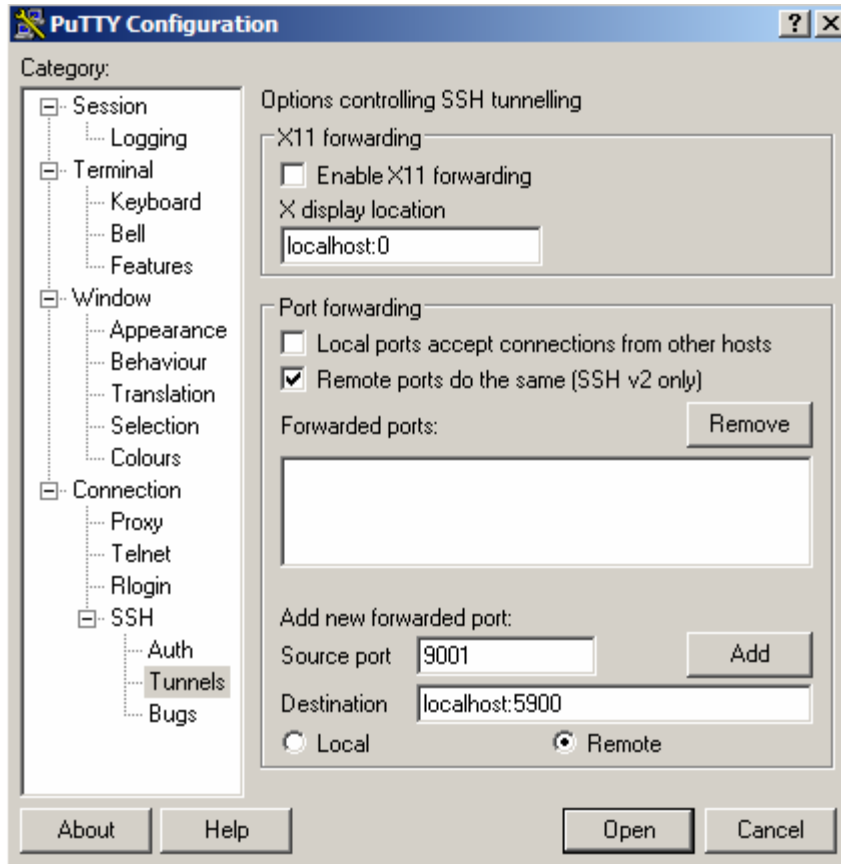


Click “Add” to complete then you could see the forwarding rule is added in the “Forwarded ports:” box, like “L9001 localhost:5900”. Then it’s done. When you

finish the setting, don't forget to go back the "Session" to save this Session Profile ("vnc client 0") by simply click the "Save". Then you could click "Open" to launch this connection. PuTTY will ask you the username and password. You can use the user account (jch280) with its password to do the authentication. After the connection is built, the tunnel of Port forwarding is built in the same time. You could test your Forwarding setting by open a command prompt windows and try to "telnet localhost 5900", see if the connect is redirected to "sparky.cs.nyu.edu port 9001".

For the "Remote Forwarding":

Suppose you wish to connection to a machine: "sparky.cs.nyu.edu" with the user account "jch280", and do the Port Forwarding as: Forward all data which goes into port 9001 in sparky.cs.nyu.edu back to you local port 5900 in you local machine. You could do this by starting create a "Session Profile" in PuTTY, let's call it "vnc server 0". Then click on the "Tunnels" (under "Connection" -> "SSH") in the left hand side menu. Check "Remote Ports do the same [SSH2 only]" and input the associated Port Forwarding data. Remember to check the "Remote" in the bottom to indicate that this is a "Remote Forwarding" setting:



Click “Add” to complete. Then you could see the forwarding rule is added in the “Forwarded ports:” box, like “R9001 localhost:5900”. Then it’s done. When you finish the setting, don’t forget to go back the “Session” to save this Session Profile (“vnc server 0”) by simply click the “Save”. Then you could click “Open” to launch this connection. PuTTY will launch a connection window and ask you the username and password, and it’s done.

You could test your Forwarding setting by open a command prompt windows and try to “telnet sparky.cs.nyu.edu 9001”, see if the connect is redirected to “localhost port 5900”.

For more information, please reference:
<http://the.earth.li/~sgtatham/putty/0.53b/html/doc/>

There are many SSH Clients which can achieve the same functionality of “Local Forwarding” and “Remote Forwarding”. For more information, please reference:

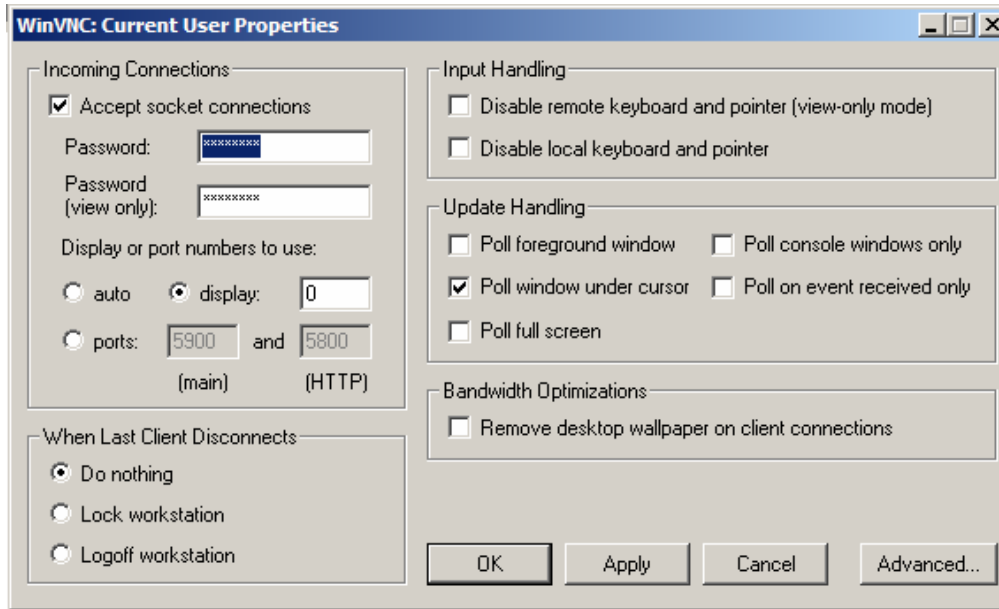
<http://www.oreillynet.com/pub/a/wireless/2001/02/23/wep.html>

A4. How to use VNC software (RealVNC and TightVNC)

The VNC (Virtual Network Computing) software is used as a “remote desktop control tool”. You install the VNC Server in your server machine and VNC Client (the VNC viewer) in the a client machine, then in the client machine you could open the remote desktop of the server to do the remote control. You could download VNC software (both server and viewer) from the web site of VNC (<http://www.vnc.com>). There are also many other VNC software modification versions providing exactly the same functionality but having some revisal like the transmitted data encoding algorithm, for example TightVNC, <http://www.TightVNC.com>, is one of them. You could choose any one of them. Here we use TightVNC as example to demo how to setup basic VNC server and viewer.

(1).The VNC Server

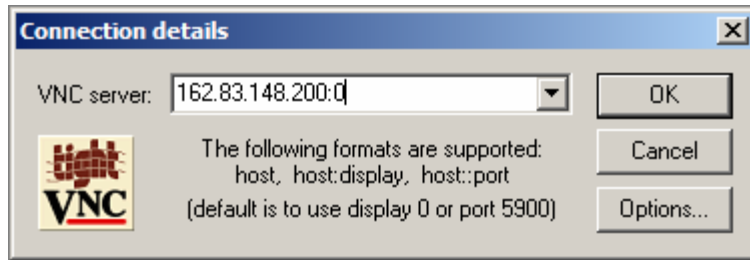
Simply download the VNC software from the VNC web site, install it. Then you could run the VNC Server by Launch the program. Remember to assign a password to you VNC Server. In the VNC property configuration, you could assign the “display number”. A display number of “0” means the server is listening on port 5900, and “1” means on port 5901, so on and so forth. This setting information is important for the VNC Client (the viewer) to connect to. The following example shows the configuration of the VNC Server running on a machine with IP address: 162.83.148.200, with the display number set to “0”:



(2).The VNC Client (the Viewer)

In the Client side, simply download the VNC software from the VNC web site, install it. Then you could run the VNC Client by Launch the viewer program. First it will ask you the address (of VNC Server) to connect to. The format to input is “address:display_number” where “address” is the Hostname or IP Address of your VNC Server, and “display_number” denotes to the display you set in the VNC Server. Then you will be asked to type the VNC Server password to authenticate.

You can install the VNC Client software in any machine, and connect to the Server (in our example, the server is the machine with IP Address: 162.83.148.200). The following is an example of running VNC Viewer to connect a VNC Server (whose IP Address is 162.83.148.200, and its display number is set to “0”, which means the VNC Server is listening in Port 5900). After the authentication, there will be a popup window whose content is the current desktop of the VNC Server.



(3).The “loopback connection” ability

The above example is the typical use of the VNC software. But the limitation is that both VNC Server and Client should be resided in the public Internet (or at least they reside inside a same LAN and can reach each other). In our project implement, we need to use the VNC Viewer to connect to a VNC Server which is listen on the localhost. This requires the VNC Server to have the ability to allow “loopback connection”. Below we introduce how to configure the VNC Server to allow loopback connection.

WinVNC Server:

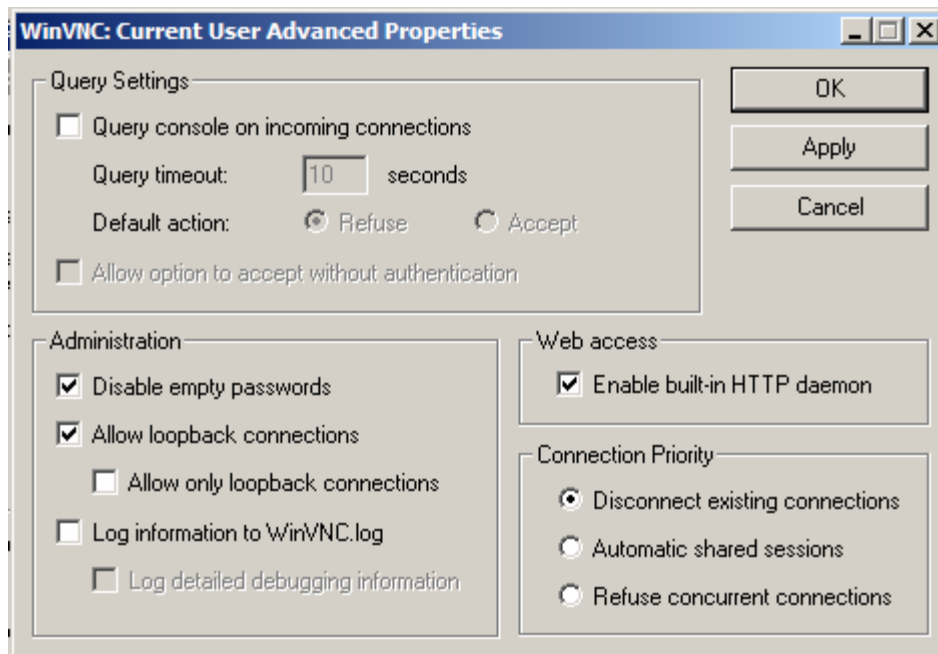
Notice that, if you install the WinVNC Server in the Microsoft Windows, by default it doesn't allow you to do the loopback connection, which means you can't use another VNC viewer to connect back to a VNC Server in localhost. This conflicts our goal of apply SSH tunneling in the VNC application. So you need to edit the register key in Windwos (simply by runing the **regedit.exe** program). Add the following register key “**AllowLoopback**” under the directory:

```
HKEY_LOCAL_MACHINE\Software\ORL\WinVNC3\
```

AllowLoopback is “Dword” type and you should set its value to 1.

TightVNC:

If you use some other modification of VNC software (e.g. TightVNC from <http://www.tightvnc.com/>) then there is an advanced setting to let you configure the local loopback connection. This way you don't need to edit the Windows Registry. Click the “Advanced...” in the VNC Server configuration window, then there will be a option to let you configure the loopback connection:

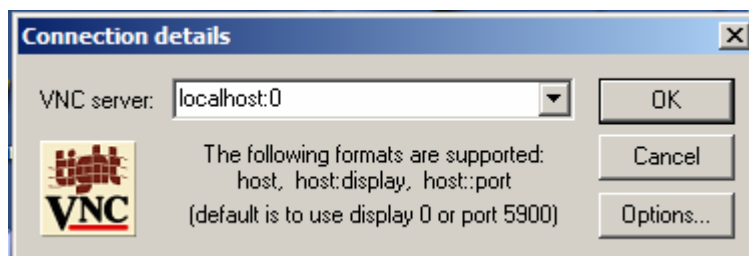


Unix:

If you are using a VNC Server in UNIX operating system, then you don't need to worry about this, by default it allows loopback connection.

In the VNC Client side:

For example, In our jttt project, we redirect the TCP connection via SSH Port Forwarding functionality. If you run jttt Applet program in a Client Machine and set up the forwarding port to Port:5900, then when you launch the VNC viewer program in the Client machine, the input form would be look like this:



For more information, please reference:

<http://www.uk.research.att.com/vnc/winvnc.html>

Reference

[Douglas 1996] Douglas E. Comer, David L. Stevens, "Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications", 2nd edition, Prentice Hall, March 25, 1996. pp. 254.

[Kurose 2002] James F. Kurose, Keith W. Ross, James Kurose, Keith Ross, "Computer Networking: A Top-Down Approach Featuring the Internet", 2nd edition, Addison-Wesley Publishing, July 17, 2002, pp. 641.

[Painter 2002] Lee David Painter, Richard Pernavas, "*SSHTools - Java SSH Solutions*", Open source software project registered in SourceForge.net on 2002-08-24.
<http://sourceforge.net/projects/sshtools>

[RFC 793] Jon Postel, "*Transmission Control Protocol*", RFC 793, September 1981.
<http://www.ietf.org/rfc/rfc0793.txt?number=793>

jttt is an independent research project and is hosted on SourceForge.net:
<http://sourceforge.net/>

