5.3. telnet

telnet is one of the earlier protocols developed for the **Internet** back in 1969. It was originally codified in **RFC 15** (http://tools.ietf.org/html/rfc15).

telnet was not built with security in mind, the protocol is sent over the wire in plain text.

5.4. RSH

Remote Shell (**RSH**) was originally written for the **BSD** (**B**erkeley **S**oftware **D**istribution) system in 1983. **RSH** is a similar system to **telnet**. Like **telnet**, **RSH** is an insecure protocol, which is not encrypted and sends data in clear text.

5.5. Cryptography

Cryptography is about securing communications. There are two main types of cryptography:

- **Symmetric**
- **Asymmetric**.

Symmetric encryption uses a single secret **shared** key, which both parties must have to communicate. **Symmetric** encryption requires relatively less computational power.

Asymmetric encryption uses mathematically-related **public** and **private** keys to communicate. **Asymmetric** encryption requires relatively more computational power.

Plain text encrypted with a symmetric encryption method can easily be turned back into plain text using the same key. A good example of a symmetric method is a **Caesar Cipher** (http://en.wikipedia.org/wiki/Caesar_cipher).

One benefit of symmetric encryption is that it is less computationally-intensive than asymmetric encryption.

One downside of symmetric encryption is that a secure shared key exchange is difficult to attain.

Figure 5.1: Symmetric Encryption

Plain text encrypted with an asymmetric public key can only be decrypted by using the corresponding private key. You cannot decrypt the cipher text using the same key which was used for encryption.

Some of the benefits of asymmetric encryption are:

- There is no key-exchange problem.
- It uses the published public key to send secure messages.

Some of the downsides of asymmetric encryption are:

- It is more computationally-intensive than symmetric encryption.
- Verifying the public key belongs to the proper party.

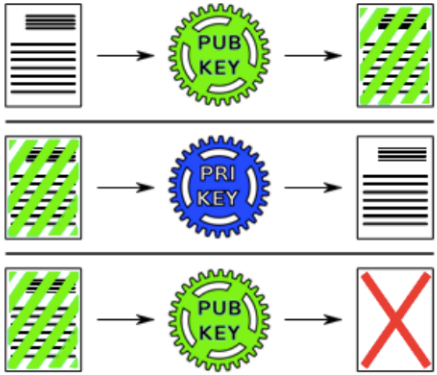

Figure 5.2: Asymmetric Encryption

By using a combination of asymmetric and symmetric encryption, you can overcome the problems associated with both. Figure 5.3 gives you an example of how to do it.

1. Party One creates a session key using a symmetric algorithm.
2. Party One then encrypts the session key, using the public key of Party Two, and sends the encrypted session key to Party Two.
3. Party Two uses their private key to decrypt the session key.
4. Both parties now communicate using the symmetric session key for encryption.

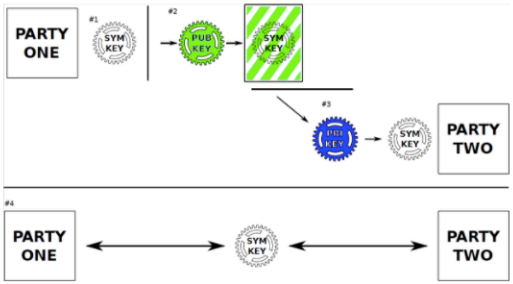

Figure 5.3: Combined Asymmetric and Symmetric Encryption

The **S**ecure **Sh**ell (**SSH**) protocol was developed to overcome the security concerns of **telnet** and **rsh**. The most widely used version of the **SSH** protocol is **OpenSSH**. **OpenSSH** is built upon the concepts of symmetric and asymmetric encryption.

The multiple layers of the **OpenSSH** architecture handle different parts of connection:

- **Transport Layer**:
  Deals with the initial key-exchange and setting up a symmetric-key session.
- **User Authentication Layer**:
  Deals with authenticating and authorizing the user accounts.
- **Connection Layer**:
  Deals with the communication once the session is set up.

An **SSH** session starts with the **Transport Layer**, which sets up the **Connection Layer**. **SSH** communications then happen over the **Connection Layer**.

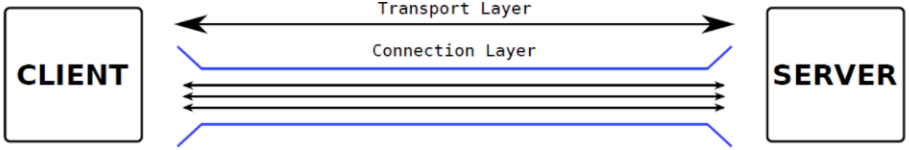

Figure 5.4: An SSH Session

5.11. OpenSSH Server

The **OpenSSH** server is configured by editing the `/etc/ssh/sshd_config` file.

Some of the commonly changed server configurations are:

- Disable root access, or allow only key-based root access, using the `PermitRootLogin` token.

  - No **root** access: `PermitRootLogin no`
  - Key-only **root** access: `PermitRootLogin without-password`

- Disable or enable **X11 Forwarding** using the `X11Forwarding` token.

```
X11Forwarding no
X11Forwarding yes
```

- Disable or enable authentication forwarding using the `AllowAgentForwarding` token.

```
AllowAgentForwarding yes
AllowAgentForwarding no
```

5.12. OpenSSH Client

The **OpenSSH** host-wide client configuration is `/etc/ssh/ssh_config`. The per-user client configuration is `$HOME/.ssh/config`. **SSH** uses a key-based authentication.

The syntax for the client configuration can be found using the command **man 5 ssh_config**.

Other protocols can be tunneled over **SSH**. The **X11** protocol support is part of the **OpenSSH** client. The **VNC** protocol support is a part of many different **VNC** clients.

You can also manually open a connection for any other protocol using the `LocalForward` and `RemoteForward` tokens in the **OpenSSH** client configuration.

`$HOME/.ssh/config` can be set up with shortcuts to servers you frequently access. Advanced, user-specific options are also available. The following is an example of how you can use **`ssh web`**:

```
Host web
     HostName www.example.com
     User webusr
```

This is an example of a more advanced configuration:

```
Host web
     KeepAlive yes
     IdentityFile ~/.ssh/web_id_rsa
     HostName www.example.com
     Port 2222
     User webusr
     ForwardX11 no
```

You can find all of the possible options in the **`ssh_config` man** page:

```
$ man 5 ssh_config
```

OpenSSH client key-based authentication provides a passwordless authentication for users. Private keys can be encrypted and password protected.

- The **ssh-agent** program can cache decrypted private keys.
- The **ssh-copy-id** program can copy your public key to a remote host.

To generate a user key for **SSH** authentication, use:

```
$ ssh-keygen -f $HOME/.ssh/id_rsa -N 'supersecret' -t rsa
```

To start **ssh-agent** and use it to cache your private key, use:

```
$ eval $(ssh-agent)
```

```
$ ssh-add $HOME/.ssh/id_rsa
```

To copy your public key to the remote system **overthere** for remote user **joe**, use:

```
$ ssh-copy-id joe@overthere
```

Consult the **man** pages **ssh-keygen**, **ssh-add** and **ssh-copy-id** for details.

There are three types of tunnels available in **OpenSSH**.

The **local tunnel** (**-L**) indicates which port is to be opened on the local host (**4242**) and the final destination to be, **charlie:2200**. The connection to the final destination is going to be made by the machine **bob**.

The **remote tunnel** (**-R**) requests machine **bob** to open a listening port **2424** to which any connection will be transferred to the destination, **charlie:2200**.

The third type of tunneling (**-D**), **Dynamic Port Forwarding,** can be found in the **ssh man** page.

Option **-N** sets the option to **not** execute a command on connection to the remote system, and option **-f** informs **ssh** to go into background just before command execution.

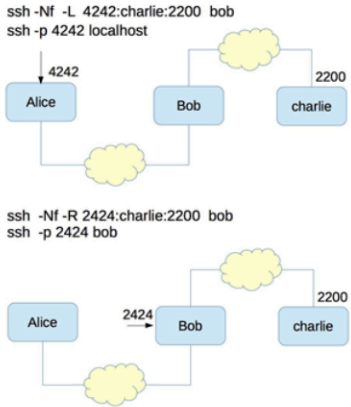

Figure 5.5: OpenSSH Tunnel

5.16. Parallel SSH Commands

Often, it is required to execute the same command on many systems to help facilitate this. The **pssh** package is available for most distributions. The **pssh** package typically includes:

- **pssh** parallel ssh
- **pnuke** parallel process kill
- **prsync** parallel copy program using **rsync**
- **pscp** parallel copy using **scp**
- **pslurp** parallel copy from hosts.

The program names may be slightly different on the different distributions.

The **pssh** command and friends use the existing **ssh** configuration. It is best to configure aliases, keys, known hosts and authorized keys prior to attempting to use **pssh**.

If there is a password or fingerprint prompt, the **pssh** command will fail.

When using **pssh**, it is convenient to create a file with a list of the hosts you wish to access. The list can contain IP addresses or hostnames. An example is provided below:

The **VNC** server allows for cross-platform, graphical remote access.

To assist in setting up a **VNC** server session, many distros provide the **vncserver** script:

- **Startup script:** `$ HOME/.vnc/xstartup`
- **Kill option:** `$ vncserver -kill <DISPLAYNUM>`

```
$ vncserver

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n
xauth: file /root/.Xauthority does not exist
xauth: (argv):1: bad display name "host:1" in "add" command
xauth: file /root/.Xauthority does not exist

New 'X' desktop is host:1

Creating default startup script /root/.vnc/xstartup
```

5.18. VNC Client

VNC is a display-based protocol, which makes it cross-platform. It also means that it is a relatively heavy protocol, as pixel updates have to be sent over-the-wire.

On its own, **VNC** is not secure after the authentication step. However, the protocol can be tunneled through **SSH** or **VPN** connections.

5.19. X Window System

The **X Window** system was developed as part of **Project Athena** at **MIT** in 1984. This simple network-transparent **GUI** system provides basic **GUI** primitives. There are no fancy desktop interfaces by default. User interfaces are provided by tool kits and add-ons.

The **X11 protocol** (the current version) is network transparent, allowing for ease of use.

5.20. X Authentication

When it comes to **X** authentication, the client-server security is done using keys. To configure which **X11** server you are using, you must set the **DISPLAY** variable.

```
$ export DISPLAY=':1'

$ xeyes &
```

X11 keys are managed using either **xhost** or **xauth**.

The **xhost** command is most useful for single-user or local-only configurations.

The **xauth** command is more flexible and allows for greater control and security.

5.21. X Tunneling

To secure the **X protocol**, it must be tunneled with **VPN** or **SSH**. **OpenSSH** supports **X11** tunneling.

To tunnel an **X11** session over **SSH** from the command line, use the following syntax:

```
$ ssh -X user@host
```

To configure your **OpenSSH** client to always tunnel the **X11** protocol, use the following syntax in your `$HOME/.ssh/config` file:

```
host foo
     ForwardX11Trusted yes
```