SSH Tunneling

Applied Cryptography

OpenSSH

https://www.tomorrowsworld.org/commentary/are-you-talking-to-yourself

- OpenSSH is a toolset for implementing SSH-enabled applications communications
- There is a complete set of tools for working with ssh keys and doing ssh encryption/decryption in the OpenSSH toolset
- OpenSSH is OSS and free, the current version is SSH2 compatible
- SSH1 is deprecated and insecure
- SSH is intended to provide an application with a secure communications channel over a potentially insecure network
- It is primarily used by a user to establish a connection for their own use, most commonly to establish a shell login to a remote system

Creating keys
SSH keys with ssh-keygen

```
ssh-keygen -t ed25519 -C "dennis on mbp"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/dennis/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/dennis/.ssh/id_ed25519.
Your public key has been saved in /home/dennis/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:KUE7q8d/i2BgdJWzxbjWfmSY+xeGUb3zGIVTyHbyk7g dennis on mbp
The key's randomart image is:
+--[ED25519 256]--+
|      . .+   . =.|
|     . o+ o   O +|
|    . =  * o o.*o|
|   . . ++.+ +..=.|
|    o o.S. + o.o+|
|   . + .  o oE+ .|
|    . =    o . . |
|     o o .. . .  |
|        o... .   |
+----[SHA256]-----+
```

puttygen for windows

- ssh-keygen will generate a keypair and store them, offering to encrypt the private key file
- ssh is a command line program that can use those key files to establish remote connections
- puttygen is a similar program for users of the putty terminal program
- websites can generate ssh keys for you but then those websites have access to your private key as well as the public one

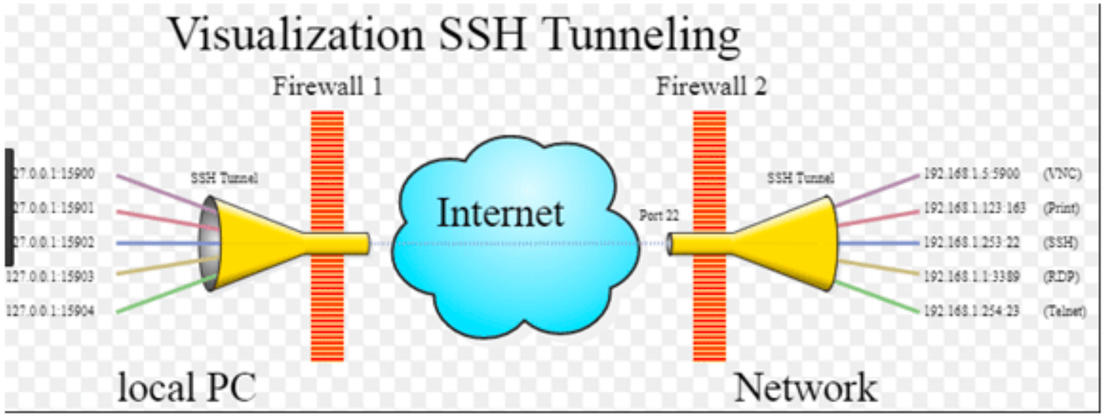
Distributing/Installing SSH Public Keys

- With SSH, a user or application will have possession of their private key, in a secure file
- Whatever service they intend to connect with should have their public key installed
- When a user creates an SSH keypair, it is for the purpose of establishing a connection, so once the keypair is generated, the public key is normally then copied to wherever it needs to be so that the desired service programs can access it
  - e.g. ssh-copy-id user@server
- An ssh-agent daemon can be used to store decrypted private keys to avoid having to continuously decrypt your private keys every time you start an application that uses SSH
- No matter how you get them there, typically the person who makes the keys puts them where they need to be for later communications, they only depend on themselves for key distribution and validation

SSH and Github

- github.com is an example of a website that can use SSH instead of HTTP, HTTPS, or other custom protocols to transfer information between the site and a user of the site
- It is a file service website that among other things allows users to maintain files that can be synchronized between local storage and the website
- It can provide unauthenticated file retrieval from the website, or it can use authentication to limit access
- It always uses authentication to provide transmission of files and other changes to the website from the client, using whatever method it used to originally retrieve the files by default
- You can use SSH to do the authentication automatically with your SSH keys instead of using login names and passwords every time you update something on the website

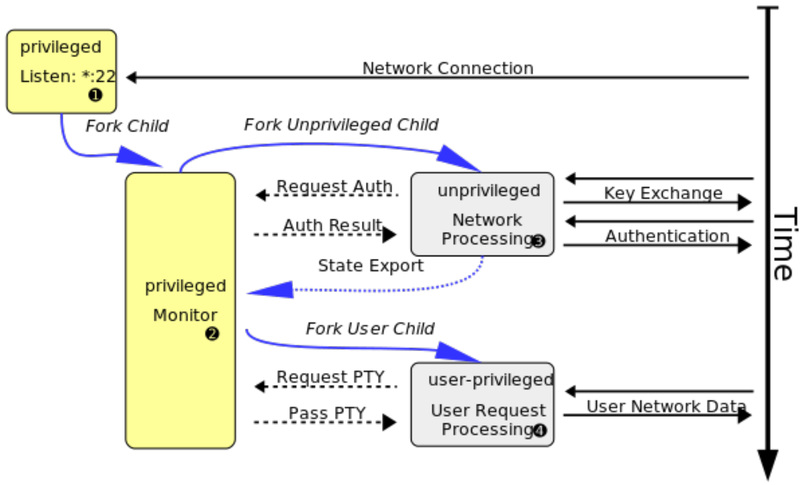
SSH Tunneling
Multiple channels



https://infosecaddicts.com/how-to-perform-dynamic-ssh-tunneling/

- SSH does not commonly use certificates but does use both asymmetric and symmetric encryption
- SSH traffic is encapsulated in the SSH protocol to provide multiple channels over a single connection
- The channels can be used for distinct purposes
- SSH therefore is a tunnelling protocol that uses encryption, not just an encrypted data stream like TLS

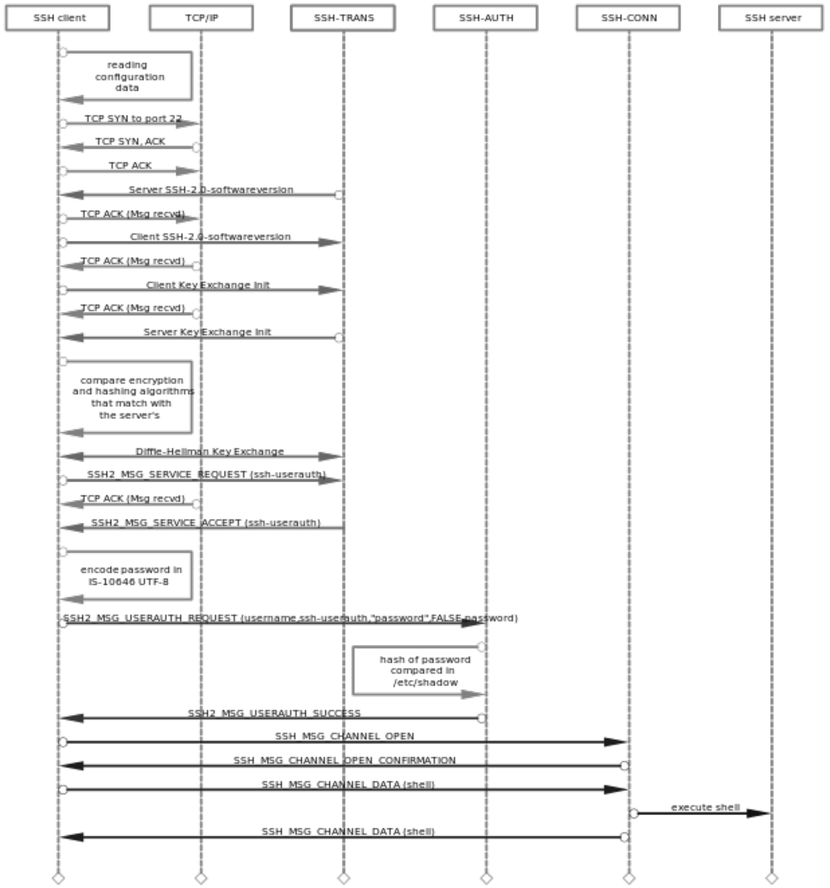
SSH Server-side Processes



https://en.wikibooks.org/wiki/OpenSSH/SSH_Protocols

- The SSH client program establishes a connection to the service daemon, which forks a monitor
- Once key exchange completes, the monitor allows the client to establish channels for application communications
- A single monitor can support multiple communications channels
- The ssh-connected login that simulates a terminal is one example of an application that can run in an SSH channel
- The use of multiple processes permits the use of unprivileged processes which limits the potential for attack/abuse

SSH Handshake



https://en.wikibooks.org/wiki/OpenSSH/SSH_Protocols

- SSH communications occurs in 3 stages
- The transport layer is established first using TCP and provides a secure transport through ssh key exchange and host key verification, then creates a session id
- The user authentication layer is next and identifies the client user for the session
- Once the user is identified, the application is given a channel to use and the application now communicates with the server, running what it needs to run and is authorized to run
- A single transport can run multiple channels

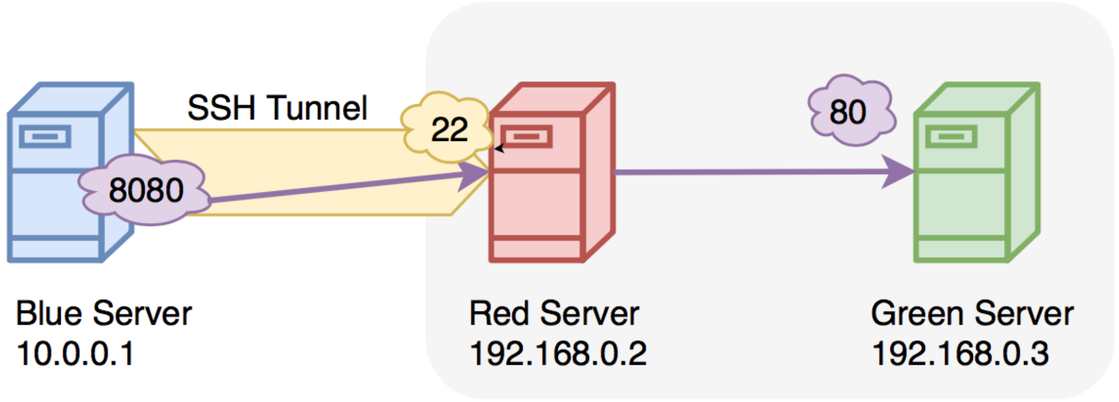
SSH Channel Uses

- Terminal access for shell access is the most common use
- File transfer using an SSH channel is better than normal FTP for all but public read-only files, the scp and sftp programs are example programs for this although you can even use a web browser to retrieve a file using sftp
- Proxy forwarding and reverse proxy forwarding is another standard use for SSH channels, but can be used to circumvent more or less all other network protections normally provided by firewalls; must be very carefully considered before being deployed
- X11 Linux window system protocol forwarding for remote graphical access is also a standard use

ssh, scp, sftp

- These command line tools are deceptively simple to use
- ssh creates a login session for a user from the local machine to a remote machine
- scp is used for file copying as is sftp
- The scp protocol version 1 has weaknesses so only use scp version 2 if you want scp
- scp behaves like cp, so it only copies files and does it simply and very fast
- sftp implements the ftp control paradigms, and can do much more than just copy files (e.g. listing directories, dealing with file name character sets, managing file permissions, etc.) which also makes it slower than scp
- scp is good for recursive copies, sftp is good for resuming interrupted transfers

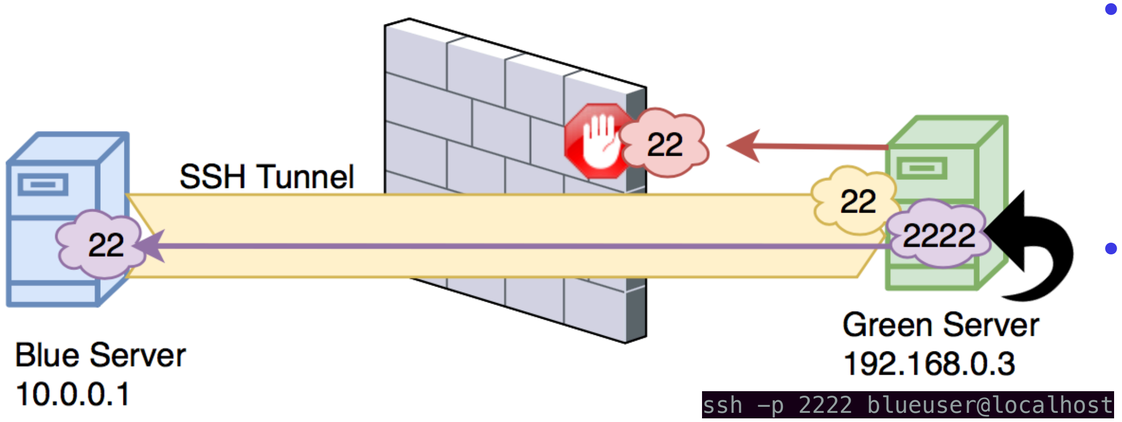
SSH Proxy



https://www.tunnelsup.com/how-to-create-ssh-tunnels/

- When you use SSH to create a tunnel for arbitrary communications carriage, you are creating a proxy
- You start the ssh program, telling it to listen on a local port, and also establish a connection to a remote ssh server which itself can then connect to a separate port local to the server's end of the connection
- Then a local program can communicate with the local port, and its communications are transported through the SSH client and server on an SSH channel and the remote end only sees a client that is local to it
- This can be used to bypass firewalls and application protections
- General format of ssh command to create a proxy
- ssh -L local-listen-ip:localport:destination-ip:destinationport [sshuser@]sshgateway-ip

SSH Reverse Proxy



https://www.tunnelsup.com/how-to-create-ssh-tunnels/

- You can use the ssh program to establish a connection to an SSH server which listens on a local port, and whatever connects to that on the server, will be connected to a port on the originating machine
- Then a remote program can communicate with a local program, and its communications are transported through the SSH client and server on an SSH channel and the remote end only sees a client that is local to it
- This can be used to bypass firewalls and application protections
- General format of ssh command to create a reverse proxy
  - ssh -R ip-on-remote-host:port-on-remote-host:destination-ip-on-localhost:destinationport-on-localhost [sshuser@]sshgateway-ip-or-name

SSH Tunnel Persistence

- The autossh command can create your SSH tunnel and keep it connected so you don't need to run the ssh command to create and recreate it all the time
  - autossh -N -i ~user/.ssh/id_ed25519 -R 2222:localhost:22 remotesshuser@remotehost
- If you set this up as a systemd service unit, you can then treat it like any other service on the machine (create a service definition file and enable/start the service) and you don't need to be logged in to use it