Connecting to OS1 for the first time

Introduction

All work in the class will be performed on the school's os1 server, which was set up specifically for this class. SSH'ing into a remote server via the command line to do remote development is a necessary skill for many CS jobs, so it's important to master this process. **This guide is written assuming you have never connected to the school's remote servers via SSH before.**

SSH stands for Secure Shell, and it is a protocol for secure remote tunneling between a client and server. In this case, we will use it to open a remote shell program (the shell is what provides the command line interface) and forward all IO to and from our local terminal window, allowing us to interact directly with the remote shell over the internet.

Behind the scenes, there is the sshd (SSH Daemon) process that runs as a service on the sever we connect to. When a client connects, sshd performs an authentication handshake, requesting credentials from the client program, before spawning a shell and establishing the tunnel. This may take the form of an interactively entered password, but there are faster and more secure methods to use. The most common alternative method uses public key cryptography, which we will set up below.

Prerequisites

To start with, you must have a terminal and the ssh command-line utility suite on your local machine. SSH is a standard, and there are different implementations, but OpenSSH is the most widely used suite of SSH tools. For mac and linux users, you should have this out of the box. For Windows users, we recommend you **install WSL (https://canvas.oregonstate.edu/courses/1870063/pages/extra-step-for-windows-users)** and use the SSH client that it provides. If you're using a different operating system than the big 3, you probably don't need help connecting via SSH, but reach out if you're having trouble! :)

Public key cryptography

To connect to the server effortlessly, we will use public key cryptography to identify ourselves to the server when logging in. To do this, we first generate a corresponding public and private key (two text files) using a key generator utility. Messages encrypted with one key can be decrypted by the other. If the server has my public key, it can encrypt a random message and send it to me using the public key.

I can then prove that I possess the private key by sending the decrypted message back, without ever revealing anything about my private key to the server. This is much more secure than a password, because the "master secret" is never sent to the server. This allows me to use the same private and public key to identify myself on different servers without ever giving the server the secret needed to identify myself. If I tried the same thing with a password, one compromised server could reveal my password for every server!

Generating a key pair

To generate the key pair, we can use a tool called `ssh-keygen`, the OpenSSH authentication key utility. We will generate a key using the (more secure than default) **ed25519 elliptic curve scheme (https://en.wikipedia.org/wiki/EdDSA)** by entering the following command on your local machine:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -N 'PASSWORD'
```

Notice the arguments we pass to the program. Arguments are words separated by whitespace following the program name. Quotes around an argument prevent it from being split if it contains whitespace characters. Programs take two kinds of arguments: positional and optional. Positional arguments are expected in a specific order after the program name, while optional arguments can be omitted or included and appear in any order. Options begin with the dash (-) character and often take an argument, themselves. We will talk more later about shell grammar, but for now, we are passing three options, each taking a single argument. In this case, the -t option specifies the cryptographic method, the -f option specifies the location to store the key files, and the -N option specifies a password to encrypt the key files.

The password specified here is used to encrypt the key files. This makes it so that if someone steals your computer or reads the files somehow, they can't use the keys without also having the password. You'll have to enter this password to decrypt the key file the first time you use it in a session (the decrypted key will be stored securely in memory so it can be reused for initiating a new connection without reentering the password, but logging off/rebooting will require you to enter the password again). You can optionally leave the password blank (keep the single quotes), which will store your keys unencrypted on your hard disk. This makes connecting faster and easier, but you should only do this if you are comfortable storing plaintext passwords on your system -- you'll need to unregister the key with OSUs servers if your computer ever becomes compromised.

Registering your keys

Now that you've generated a key pair, you want to go ahead and copy the public key to the OSU server so you can use the keys to log in. The easiest way to do this is to use the `ssh-copy-id` utility. Basically, it adds your public key as an entry to the `~/.ssh/authorized_keys` file on the remote server, creating it with the appropriate permissions set, if the authorized_keys file doesn't yet exist. The sshd service will use this key to perform authentication on future connections. Your entire home directory (the ~) is shared across the network at OSU, so registering this key on one server registers it on all servers. You can use the following command to copy over your keyfiles:

```
ssh-copy-id ONID@access.engr.oregonstate.edu
```

You will be prompted to enter your ONID password, and then authenticate with DUO. After the key has been copied over, you should be able to log in to flip without duo or a password, like so:

```
ssh ONID@access.engr.oregonstate.edu
```

You should see a message of the day and be presented with a prompt to enter a command. Press Ctrl-D on an empty prompt line to close the connection and log out.

Moving around the OSU network

Now that you've registered your local key with the server, you can log in remotely. However, you also want to be able to jump from one server to another inside the OSU network. For example, you might want to be able to jump from flip to os1. Since the home directory is shared between all the servers, this is actually pretty easy; you just create a new key pair on the server and copy the public key to the server's authorized keys file. This way whichever host you jump from will have the private key, and whichever host you jump to will have the public key in its authorized keys.

You ca this by first SSH'ing into the school server and then following all the previous steps one more time to generate a new key pair and copy it to the server (to itself, really).

Connecting to OS1

It is important to understand that OS1 has no public facing IP address, so it is only accessible from a system within OSUs local network. If you are connected to the school's VPN, or if you are connected to the on-campus network, then you can directly connect to os1:

```
ssh ONID@os1.engr.oregonstate.edu
```

Otherwise, you'll need to use another public-facing server as a *jump host.* Requiring users to connect to internal servers through an external *Bastion* server is a common setup that enhances network security by forcing all external traffic to pass through a single point. At OSU, the designated server for you to use for this is access.engr.oregonstate.edu, which will land you in one of three servers after login: flip1, flip2, or flip3. From here you can then connect to the os1 server. This is called a Proxy Jump or Jump Host configuration, and ssh allows you to specify jump hosts in a single command:

```
ssh ONID@os1.engr.oregonstate.edu -J ONID@access.engr.oregonstate.edu
```

First, ssh will establish a connection to flip via the jumphost, access.engr.oregonstate.edu, and then it will tunnel a second connection to os1.engr.oregonstate.edu through the first connection. This is one of the reasons we generated a second keypair on the servers, so we could use access.engr.oregonstate.edu as a jumphost. Go ahead, try it out! You should see two message of the day printouts, first for flip, then for os1.