SSH

Concepts

SSH - Secure SHell Overview

How SSH Works

How SSH Authenticates Linux Users

Verify EC2 instance Key-Pairs

Generating an SSH Key Pair

Custom SSH Key Pair for EC2

scp – Secure Copy

Concepts

Encryption : Scrambles information so other people can't read it.

1. Shared secrets

- Common case. There is a "password" (or similar) that both sides must know to encrypt/decrypt.

2. Public keys

- It contains a pair of keys. Things encrypted with one may only be decrypted with the other. Usually, one is kept secret while the other is publically distributed.
  - **a. Public key authentication**
    If we have the public key, we can use it to check if the other end really has the private key (thus proving its authenticity).
  - **b. Fingerprints**
    When we receive a public key, we may not know if it really belongs to the person we want to talk to. One method of verifying keys is via fingerprints. If we know the key's fingerprint ahead of time, we can check that against the key we have. (Fingerprints are much shorter than actual keys and more practical to write down on sheets of paper.)

Concepts

Accessing machines remotely.

1. Local Access

- When you sit at a computer (and a command line), everything you type goes directly to the shell (which provides the "command line").

2. Telnet

- telnet goes across the network to simulate the same thing. Whatever you type is picked up, sent across the network, and sent to the shell on the remote machine.
- **telnet is plaintext :** telnet sends everything literally across the network, so anyone watching the network will see exactly what you type, including things that you don't even see on the screen (like passwords).

3. Ssh

- ssh has the same basic concept as telnet--it takes what you type and sends it across the network to a shell on the remote computer.
- **ssh is encrypted :** ssh, however, uses encryption to protect your information. People watching an ssh session on the network will see some garbage information.

SSH - Secure SHell Overview

- **SSH** stands for "**S**ecure **SH**ell"
- The most common way of connecting to a remote Linux server is through SSH.
- SSH stands for **Secure Shell** and provides a safe and secure way of executing commands, making changes, and configuring services remotely.
- ssh is secure in the sense that it transfers the data in encrypted form between the host and the client.
- When we connect through SSH, **we log in using an account that exists on the remote server.**

How SSH Works

- When you connect through SSH, you will be dropped into a **shell session**, which is a text-based interface where you can interact with your server.
- The SSH connection is implemented using a **client-server model**.
- For the duration of your SSH session, any **commands that you type** into your local terminal are **sent through an encrypted SSH tunnel** and **executed on your server**.
- This means that for an SSH connection to be established, the remote machine must be running a piece of software called an **SSH daemon**. (***systemctl status sshd***)
- This software listens for connections on a **specific network port(default 22)**, authenticates connection requests, and spawns the appropriate environment if the Linux user provides the correct credentials.

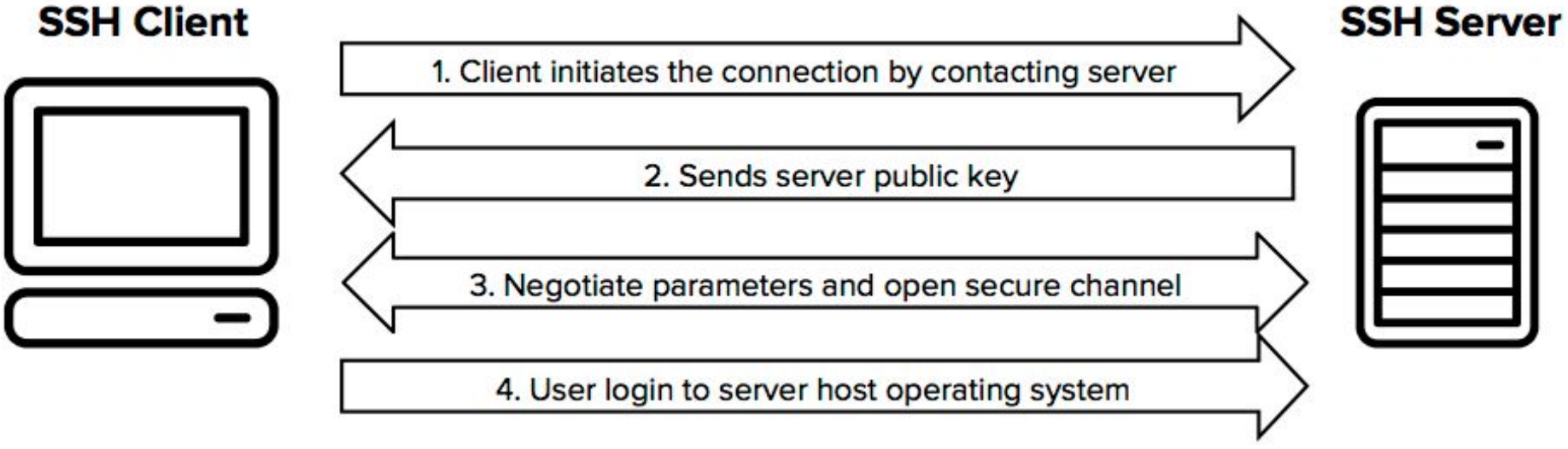
Sshd daemon

- Amazon Linux 2 - EC2 Instance SSH Daemon Status
- On EC2 instance, sshd daemon is **enabled** (Starts the daemon on Instance **start/reboot**

```
[ec2-user@ip-172-31-27-131 ~]$ systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-02-06 09:27:37 UTC; 5s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 879 (sshd)
   CGroup: /system.slice/sshd.service
           └─879 /usr/sbin/sshd -D

Feb 06 09:27:37 ip-172-31-27-131.ap-south-1.compute.internal systemd[1]: Starting OpenSSH server daemon...
Feb 06 09:27:37 ip-172-31-27-131.ap-south-1.compute.internal sshd[879]: Server listening on 0.0.0.0 port 22.
Feb 06 09:27:37 ip-172-31-27-131.ap-south-1.compute.internal sshd[879]: Server listening on :: port 22.
Feb 06 09:27:37 ip-172-31-27-131.ap-south-1.compute.internal systemd[1]: Started OpenSSH server daemon.
```

How SSH Works



- The user's computer must have an **SSH client** (*Putty / Git Bash / Linux Terminal/ MobaXterm*)
- This is a piece of software that knows how to communicate using the SSH protocol and can be given information about the remote host to connect to, the Linux username to use, and the credentials that should be passed to authenticate.

How SSH Authenticates Users

- Clients generally authenticate either using passwords (less secure and not recommended) or SSH keys, which are very secure.
- Password logins are encrypted and are easy to understand for new users.
- However, automated bots and malicious users will often repeatedly try to authenticate to accounts that allow password-based logins, which can lead to security compromises.
- For this reason, a recommended approach is setting up SSH key-based authentication for most configurations.
- SSH keys are a matching set of cryptographic keys which can be used for authentication. Each set contains a **public and a private key**.
- The public key can be shared freely without concern, while the **private key must be vigilantly guarded and never exposed to anyone.**

How SSH Authenticates Linux Users

- To authenticate using SSH keys, a user must have an **SSH key pair** on their local computer. On the remote server, the public key must be copied to a file within the Linux User's Home directory at **~/.ssh/authorized_keys**.

  $ cat /home/ec2-user/.ssh/authorized_keys

- This file contains a **list of public keys**, **one-per-line**, that are authorized to log into this account.
- When a client connects to the host, wishing to use SSH key authentication, it will inform the server of this intent and will tell the server which public key to use. The server then checks its **authorized_keys file for the public key**, generates a random string, and encrypts it using the public key. This encrypted message can only be decrypted with the associated private key. The server will send this encrypted message to the client to test whether they actually have the associated private key.
- **Note: A directory starting with dot "." in Linux is a hidden directory.**

How SSH Authenticates Linux Users

- Upon receipt of this message, the client will decrypt it using the private key and combine the random string that is revealed with a previously negotiated session ID.
- It then generates an MD5 hash of this value and transmits it back to the server.
- The server already had the original message and the session ID, so it can compare an MD5 hash generated by those values and determine that the client must have the private key.

Command for ssh:

ssh -i key.pem ec2-user@<PUBLIC_IP> -vv

- Here **-vv** is verbose mode of ssh command to display steps executing in background to setup a ssh connection.

Verify EC2 instance Key-Pairs

- The Private Key that is downloaded while launching EC2 instance, the public key of this private key is added in this file on the EC2 instance **/home/ec2-user/.ssh/authorized_keys**

  $ cat /home/ec2-user/.ssh/authorized_keys

  To check public key using private key file, use below command in Local Git Bash.

  $ ssh-keygen -y -q -f private-key.pem

  OR Check using puttygen , load private key file, public key will displayed.

- Here, the output of above command and content of **authorized_keys files** on EC2 instance would be same.
- **<u>This indicates, you can generate public key using private key, vice-versa will NEVER work.</u>**

Generating an SSH Key Pair

- To generate an RSA key pair (public and private) on your local computer, type in **Git Bash**:

  $ **ssh-keygen**

- Leave all options default and press **Enter**

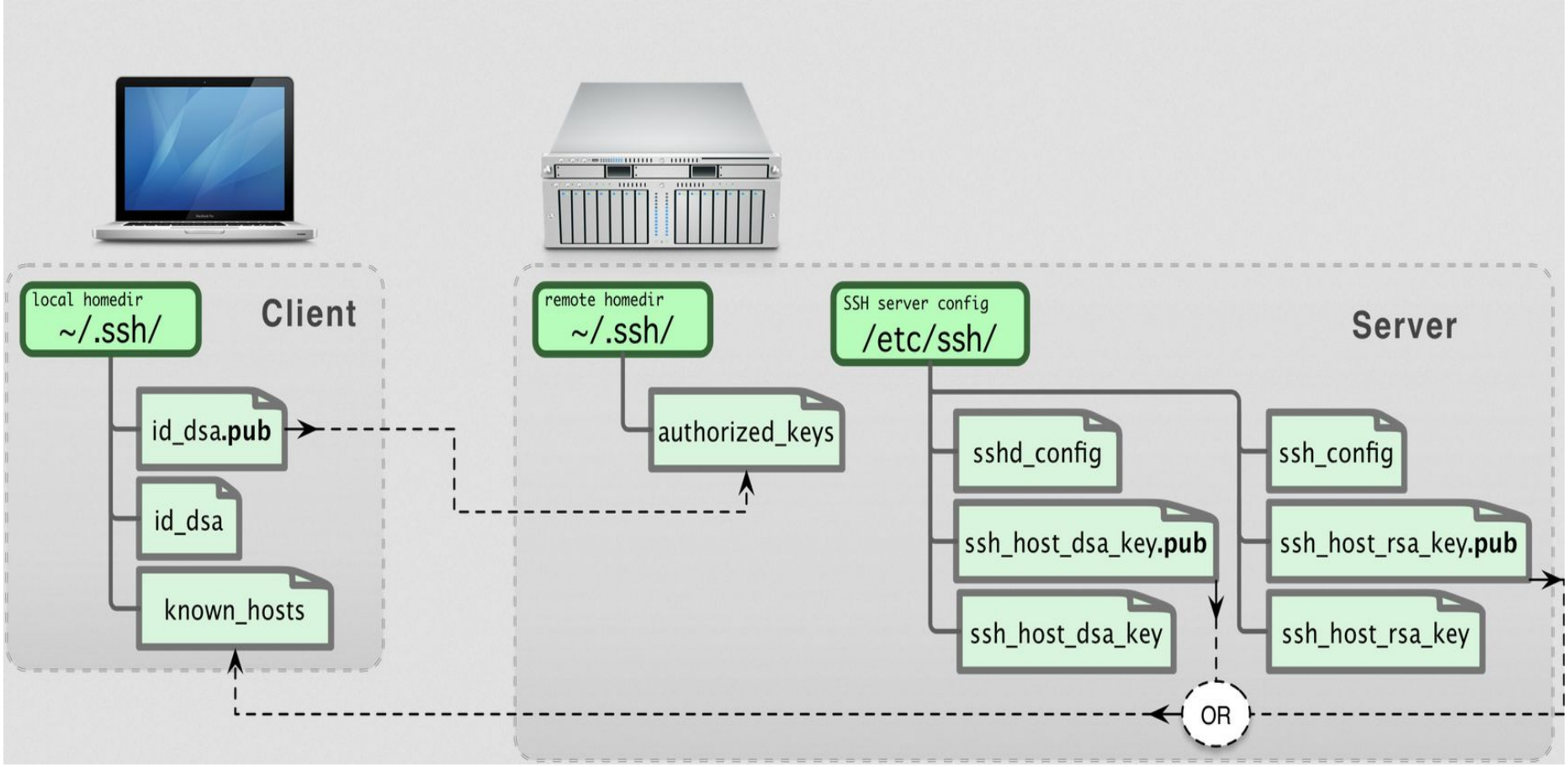
  ~/.ssh/id_rsa: Private Key File. DO NOT SHARE with anyone not intended to login using ssh

  ~/.ssh/id_rsa.pub: The associated public key. Can be shared.

- The private key must remain hidden, while the public key must be **appended to the remote host under /home/ec2-user/.ssh/authorized_keys**
- After copying the public key to the remote host the connection will be established using SSH keys and not the password.
- To check content of public key using private key file
- **id_rsa =>** Private Key File
- **id_rsa.pub =>** Public Key File

  $ ssh-keygen -y -f -q id_rsa

How SSH Works



Custom SSH Key Pair for EC2 Linux User

```
sudo useradd newuser
mkdir /home/newuser/.ssh/
chmod 700 /home/newuser/.ssh/
vi /home/newuser/.ssh/authorized_keys
chmod 600 /home/newuser/.ssh/authorized_keys

# Use Command for login from local machine
chmod 400 id_rsa
ssh -i id_rsa newuser@<PUBLIC_IP>
```

- Create a Linux User
- Create a directory .ssh inside users home directory:
- Modify permissions of .ssh directory
- Create a authorized_keys file under .ssh directory. Add/Append the **id_rsa.pub** public key file content in `/home/newuser/.ssh/authorized_keys`
- Modify permissions of **authorized_keys** file under **.ssh** directory
- On **Local Client Machine**,
- Once above file changes are done, use the command to login using ssh with associated private key for above configured public key.

Note :

- **.ssh** directory permissions should be “**drwx------**”, with owner as Linux user.
- **authorized_keys** file permissions should be “**rw-------**”, with owner as Linux user.
- **id_rsa** -> Private Key

Custom SSH Key Pair for EC2 Linux User

- In case for existing Linux user i.e **ec2-user** , a new key needs to be configured, **append** the public key (**id_rsa.pub** file) into **/home/ec2-user/.ssh/authorized_keys** file
- Modify the permissions of the private key file locally and use the below command with respective key pair with private key for above configured public key.
- *$ ssh -i id_rsa ec2-user@<PUBLIC_IP> -vv*

EC2 Instance Recover access

- Below is a summary of the steps required and covered in this post to recover access to your EC2 Instance after losing your key-pair.
  - Power off the original(**target**) EC2 instance of which you want to regain access.
  - Launch new (**recovery**) EC2 instance by generating new ssh key-pair
  - Login via ssh to the new recovery instance
  - Detach the primary EBS volume from original (**target)** instance
  - Attach/Mount the previously detached volume to the new (**recovery**) instance
  - Copy **authorized keys** from recovery instance to the mounted (target) volume
  - Unmount target volume from recovery instance and re-attach back to original (**target)** instance.
  - Start the original (**target**) instance and login with new key-pair
  - Delete temporary(recovery) instance

Linux User Accounts on EC2

- Each Linux instance launches with a default Linux system user account. The default user name is determined by the AMI that was specified when you launched the instance.
- For **Amazon Linux 2** or the Amazon Linux AMI, the user name is **ec2-user**.
- For a CentOS AMI, the user name is **centos** or **ec2-user**.
- For an **Ubuntu AMI**, the user name is **ubuntu**.

scp – Secure Copy

- ssh also has a program for copying files across the network.
- It encrypts everything, of course, so neither your password nor the data is visible to anyone on the network.
- **scp (secure copy)** command in Linux system is used to copy file(s) between servers in a secure way.

$ scp [options] <SRC_FILE_PATH> <DEST_FILE_PATH>

Local to Remote Secure Copy:

$ scp -i key.pem -v file.txt ec2-user@hostname:/home/ec2-user/

Remote to Local Secure Copy:

$ scp -i key.pem -v ec2-user@hostname:/home/ec2-user/file.txt ./

Remote SSH Commands

- *$* ***ssh -i <private_key.pem> ec2-user@<SERVER_IP>*** *"<linux command>"*
- ssh user@server1 '**df -H**'
- ssh user@server1 '**sudo useradd user1**'
- ssh user@server1 '**date**'
- ssh user@server1 '**uptime**'
- ssh user@server1 '**uptime && date**'