# CIS 444 - SSH Report

Lucas Hasting

September 2024

# Contents

# 1 What is SSH?

SSH (Secure Shell) is a protocol to access a shell securely. This shell is connected through a Client account and a Server account, and the connection is encrypted using an asymmetric encryption protocol.

An SSH session is established in two stages:

## 1.1 Stage 1

The first stage is to agree upon and establish encryption to protect future communication, which is done in the following steps:

CLIENT:

```
1. Establish a TCP connection
2. Receive protocol version from SERVER
3. If(cannot match protocol): terminate
4. Receive public host key from SERVER
5. If (unintended public host key): terminate
6. Negotiate session key with SERVER using the Diffie-Hellman Algorithm
```

SERVER:

```
1. wait for TCP connection from CLIENT
2. Send protocol version that is supported
3. if(no more connection): go to 1
4. Send public host key to CLIENT
5. if(no more connection): go to 1
6. Negotiate session key with SERVER using the Diffie-Hellman Algorithm
```

The connection is then secured using the binary packet protocol which uses symmetric encryption with the session key as the key.

The session is now in a secured tunnel between CLIENT and SERVER.

Note: The public host key determines if the Client is the intended target of the Server for the session.

## 1.2 Stage 2 - Using Password Authentication

Note: A connection is already established when this stage is entered

CLIENT:

```
1. Receive prompt from SERVER, and enter for username and password
2. Send username and password to SERVER
3. if(success): Receive Shell
4. if(fail): go to 1
```

SERVER:

```
1. Send prompt (for username and password) to CLIENT, wait for response
2. Receive username and password
3. if (account with username and password exists): Send shell
4. if (account with username and password does not exist): go to 1
```

The connection is established from CLIENT account to SERVER account, although, this method is not suggested due to the ability to brute force the username and password.

Note: The connection is still encrypted through the binary packet protocol.

## 1.3 Stage 2 - Using SSH Key Pairs

Note: A connection is already established when this stage is entered

CLIENT:

```
1.  Send ID (for the key pair it associates with) to SERVER
2.  if(loose connection): terminate
3.  Receive message, public key from SERVER
4.  if(has private key): decrypt message
5.  if(no private key): terminate
6.  new_val = [message] [shared session key] //combined in some way
7.  hash = MD5(new_val) //MD5 hash function
8.  Send hash to SERVER
9.  if(success): Receive Shell
10. if(loose connection): terminate
```

SERVER:

```
1.  Recieve ID from CLIENT
2.  Check authorized_keys to see if ID exists
3.  if(ID does not exist): restart
4.  message = random_number() //generate some random number
//the public key is associated with ID
5.  Send message, public key to CLIENT
6.  if(loose connection): restart
7.  Recieve hash from CLIENT
8.  new_val = [message] [shared session key] //combined in some way
9.  server_hash = MD5(new_val) //MD5 hash function
//HMAC - see section 2.2
10. if(hash == server_hash): Send Shell
11. if(hash != server_hash): restart
```

This method tries authentication only once and does not allow for inputs to determine authentication, preventing a brute force attack.

Note: The connection is still encrypted through the binary packet protocol.

## 1.4 Avoiding the Man in the Middle

A man-in-the-middle attack occurs when an attacker gets in between two channels (like a Server and a Client) and communicates to each of them like they are the other party.

Before using SSH, SSH offers to generate a hashed fingerprint, this fingerprint will be used to identify that the Client truly is the Client and not anyone else (It will be used as the Public Host Key). The file that contains the fingerprint is located at /.ssh/known_hosts, if a man-in-the-middle attack occurs, the fingerprint will not match the Client fingerprint and a warning will be issued.

SSH also protects against the man-in-the-middle when using key pairs, as only the Client has the private key and that cannot be replicated, so only the Client can decrypt messages sent by the Server.

# 2 SSH Processes

## 2.1 Symmetric/Asymmetric Encryption

In Symmetric encryption, a single key is used for encryption and decryption, allowing for anyone with the key to understand the message being sent.

In Asymmetric encryption, two keys are used, one for encryption and one for decryption. The two possible keys are a private key and a public key. A private key is something only one person knows, and a public key is something everyone knows. The public key is used for encryption, while the private key is used for decryption.

## 2.2 Hashing

A hash function is a function that takes in some input and outputs a "unique" number of a specific size, and the output "cannot be reversed."

The content in quotes is the idea of what a hashing function is, but it does not hold true for a lot of the algorithms.

A hash-based message authentication code (HMAC) is a message code that is hashed to determine the authenticity of the code (used to make sure a code is unmodified). This is used by SSH using key pair authentication.

The binary packet protocol makes use of a message authentication code (MAC) algorithm which uses HMAC, the binary packet protocol itself it quite complicated so it will not be discussed.

## 2.3 Diffe-Hellman

The Diffie-Hellman algorithm allows for an identical session key to be made from public and private data. The algorithm is shown below:

CLIENT:

```
// p is a prime number, g is a primitive root
1. communicate with SERVER to agree on values p and g
// a is an integer
2. a = some secret value (unknown to SERVER)
3. A = g^a % p
4. Send A to SERVER
5. Receive B from CLIENT
6. s = B^a % p
```

SERVER:

```
// p is a prime number, g is a primitive root
1. communicate with CLIENT to agree on values p and g
// b is an integer
2. b = some secret value (unknown to CLIENT)
3. B = g^b % p
4. Send B to CLIENT
5. Receive A from CLIENT
6. s = A^b % p
```

$s$ is the session key used by the binary packet protocol.

# 3 The Process

In this demonstration, OpenSSH will be configured and used with a Client and Server to allow a secure connection from a Client account to a Server account. The Server account will be accessed via a Shell provided by OpenSSH. See sections 1 and 2 for a high-level description on how SSH works. The OS chosen for this lab is Debian, so the commands used are subject to change based on the OS being used.

## 3.1 Part 1 - Server Installation

First, get the Server and Client installed on some machine (it can be a virtual machine). Once installed, open the terminal (if it has a GUI) and type the following command to update the list of packages that can be installed:

```
sudo apt-get update
```

## 3.2 Part 2 - OpenSSH Installation and Activation

For all prompts, select y for yes:

To install ssh on the Server, type the following:

```
sudo apt-get install openssh-server
```

To start ssh type

```
sudo service sshd start
```

Note: By default password authentication is enabled, to ensure this open the /etc/ssh/sshd_config using nano and see if "PasswordAuthentication yes" exists, if it does not, add it to the file.

To install ssh on the Client, type the following:

```
apt-get install openssh-client
```

Note: it could already be installed on the operating system

## 3.3 Part 3 - Using SSH

To initiate the first stage of SSH, from the Client, run the following:

```
ssh <username>@<location-of-server>
```

The location of the Server can be found by using the following commands from the Server :

```
sudo apt-get install curl
curl ifconfig.me
```

If it is your first time initiating stage 1, it will ask you if you want to save a fingerprint to your .ssh/known_hosts file, type yes, it will be used as the public host key. After typing yes, view the file using nano to ensure the fingerprint is there and it matches, if not, it is a sign that there is a man in the middle.

Once initiating the first stage again, if the fingerprint is on your system, it will not ask to save another one. It will then ask for a password to the account you tried to connect to. If you give it the correct password, the second stage will be complete, otherwise, you will be given another prompt for a password.

## 3.4 Part 4 - SSH Key Generation and Key Setup

Password authentication is unsecure due to the ability to brute force the password, so, SSH key pairs can be used instead.

To generate a key pair, type the following in the Client (the ed25519 algorithm is recommended):

```
ssh-keygen -t <encryption-algorithm>
```

This command generates a public and private key, as a reminder, the public key is used for encryption, and the private key is used for decryption, because of this, the public key must be on the Server, and the private key must be on the Client.

The private key is already on the Client (it is safer to put it on a flash drive that the Client uses) so the public key needs to be moved to the Server. This can be done using a couple of methods.

### 3.4.1 Method 1 - secure copy

The public key itself needs to be in the users .ssh folder, and the scp (secure copy) command will used to place it there. The secure copy command uses ssh to copy over files, type the following to move over the public key.

```
scp /public/key/location <username>@<location-of-server>:/destination/
   location
```

Make sure the destination location is the .ssh folder located in the users home folder.

The file used to find the public key is the authorized_keys file, the public key can be renamed to that using the following command:

```
mv public_key authorized_keys
```

### 3.4.2 Method 2 - ssh copy id

This next command does everything the previous method did, but in one operation from the user's perspective (1 command), to move the private key over type:

```
ssh-copy-id -i /public/key/location <username>@<location-of-server>
```

## 3.5 Part 5 - Using SSH More Securely

The final step to using SSH more securely is to open /etc/ssh/sshd_config in nano and set "PasswordAuthentication no" and either add or edit "PubkeyAuthentication yes"

Make sure to run the following from the Server once the file is saved:

```
sudo service sshd restart
```

Now (if no problems occurred) you can connect from the Client to the Server using key pair authentication, below is a list of methods to do that:

### 3.5.1 Method 1: using the private key file

You can use the private key file itself for authentication using the following command:

```
ssh -i location/of/private/key <username>@<location>
```

However, you can also rename/move the file like with the Server , and it would allow for the following command to work:

```
ssh <username>@<location>
```

To do the second command make sure the file is in the users .ssh folder and run the following command:

```
mv <private_key_name> id_rsa
```

### 3.5.2 Method 2: using host configurations

Another method is using a host configuration, to do so, in the .ssh folder on the users home folder, open the config file in nano, and add the following information using the shown syntax:

```
Host <location>
  HostName <name>
  User <user>
  IdentityFile <location of private key>
```

Once that file is saved the following command can be used:

```
 ssh <name>
```

## 3.6 Part 6 - Important Files

/ect/ssh/ssh_config: change file location names for ssh (client) such as known_hosts

/etc/ssh/sshd_config: change file location names for ssh (server), change authentication methods, and make more configurations

.ssh: located in the user's home folder

.ssh/known_hosts: contains the public host key used for stage 1 of ssh

.ssh/authorized_keys: contains authorized public keys for the account on the Server

.ssh/id_rsa: contains the private keys for the account on the Client

.ssh/config: contains hosts for ssh