SSH Fundamentals

Basics of using **S**ecure **SH**ell

Prerequisites

- This workshop assume an understanding of
  - Asymmetric encryption concepts
  - Basic program installation skills
- Follow Along
  - Many of these slides will allow you to follow along.
  - You'll need to run the following programs from your command line
    - ssh
    - scp
    - sftp
    - ssh-keygen

What is **S**ecure **SH**ell?

- A **shell** is an execution environment.
  - Associated with a user ID on a machine
  - Defines a PATH variable for locating commands, as well as other variables.

- A **remote** shell:
  - Executes on a remote machine
  - Must be protected from
    - Unauthorized users
    - Snooping on authorized users

SSH (Secure SHell) provides a **secure remote shell** with two-way authentication, encryption of traffic, and a variety of execution methods.

Client and Server Components

- Client
  - Initiates the connection.
  - It's what most people mean when they say "ssh".
  - Many instances can run concurrently.
  - Configuration files in ~/.ssh.
    Windows can be different.

- Server
  - Listens on port 22
  - Known as sshd (ssh daemon)
  - Server configuration files in
    - `/etc/ssh/sshd_config`
    - `/etc/ssh` usually contains server keys
  - Server log in `/var/log/messages`.

Most of us only have to work with the client-side of SSH

SSH Client

- Comes standard on Linux and macOS.
- Windows (https://www.ssh.com/ssh/client)
  - OpenSSH – recommended for this workshop
  - PuTTY – long time favorite for Windows
  - WinSCP – GUI file transfer
- OpenSSH on Windows
  - Included in Windows 10 since Fall 2018. Use the Windows 10 "**Add Feature**" to add it to your path.
  - For Windows 7 and earlier Windows 10:
    - **Win32-OpenSSH** Release Page: https://github.com/PowerShell/Win32-OpenSSH/releases
    - Download `OpenSSH-Win64.zip`
    - Unzip onto your file system. This will create an `OpenSSH-Win64` directory with binaries.
    - Add new directory to your user `PATH` environment variable.

Establishing an SSH Connection

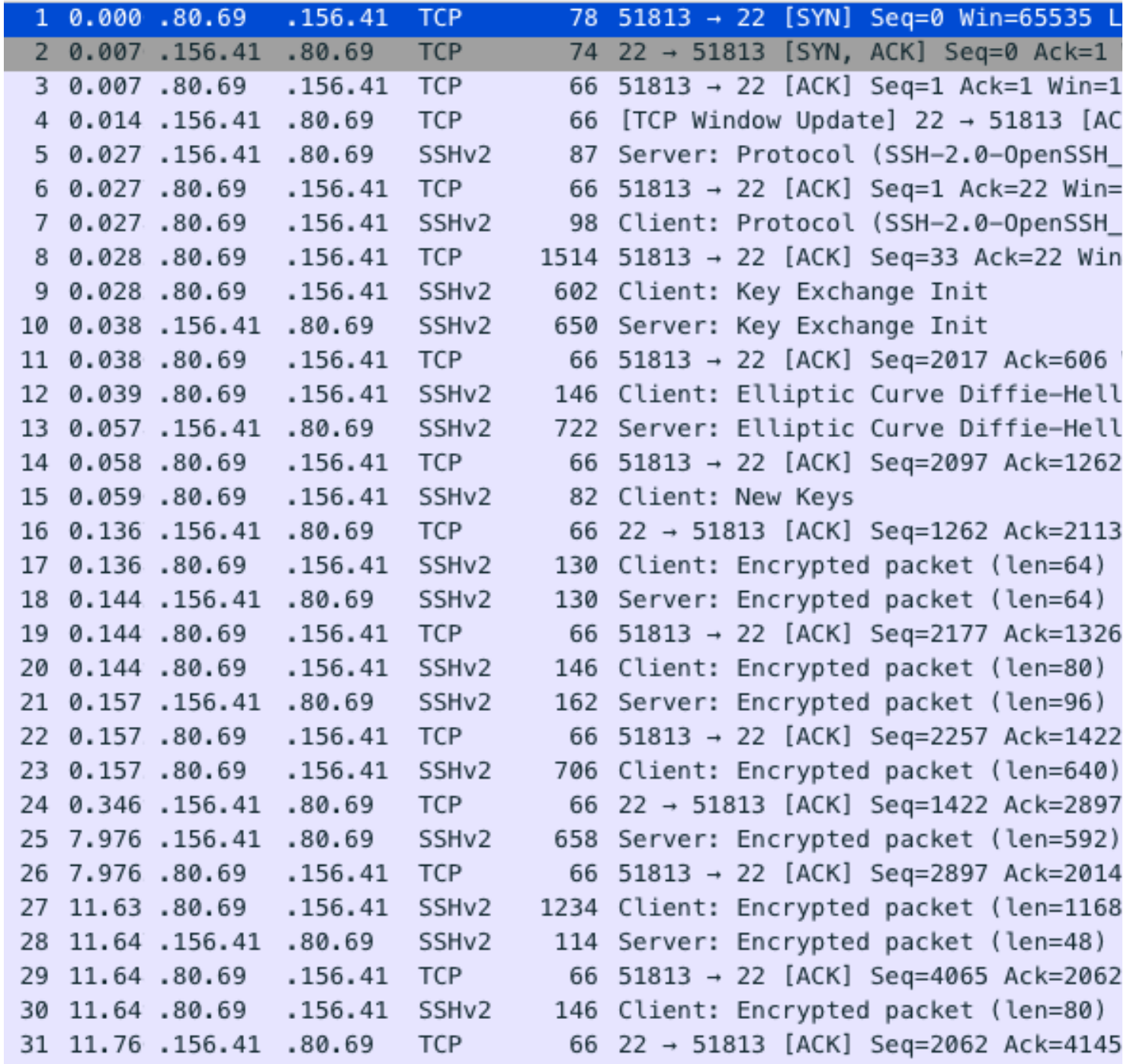
1. Establish TCP connection.
2. Handshake
   i. Client validates server's key
   ii. Client and server exchange supported cipher suites to determine compatibility.
   iii. Session key is determined.
3. Client authentication
   i. **client password** – Password is validated against account on server. User ID and password are sent only **after** encrypted tunnel is established.
   ii. **client certificate** – Certificate is validated against a list configured for the user on the server.

Note Step 2.i authenticates the server. This helps protect against server spoofing.

SSH Handshake

Packet	Description
1-3	TCP Handshake
5	Server states its protocol and version
7	Client responds likewise
9-10	Client and server exchange lists of supported algorithms and key sizes. Note the server gets the last word on options.
12-13	Client chooses (from list in 10) Elliptic Curve Diffie-Hellman. Server confirms.
15	Client sends session key.
25-26	Server requests authentication of user.
27	Client responds with certificate.

Everything after packet 15 is encrypted.



1	0.000	.80.69	.156.41	TCP	78	51813 → 22 [SYN] Seq=0 Win=65535 L
2	0.007	.156.41	.80.69	TCP	74	22 → 51813 [SYN, ACK] Seq=0 Ack=1
3	0.007	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=1 Ack=1 Win=1
4	0.014	.156.41	.80.69	TCP	66	[TCP Window Update] 22 → 51813 [AC
5	0.027	.156.41	.80.69	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_
6	0.027	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=1 Ack=22 Win=
7	0.027	.80.69	.156.41	SSHv2	98	Client: Protocol (SSH-2.0-OpenSSH_
8	0.028	.80.69	.156.41	TCP	1514	51813 → 22 [ACK] Seq=33 Ack=22 Win
9	0.028	.80.69	.156.41	SSHv2	602	Client: Key Exchange Init
10	0.038	.156.41	.80.69	SSHv2	650	Server: Key Exchange Init
11	0.038	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=2017 Ack=606
12	0.039	.80.69	.156.41	SSHv2	146	Client: Elliptic Curve Diffie-Hell
13	0.057	.156.41	.80.69	SSHv2	722	Server: Elliptic Curve Diffie-Hell
14	0.058	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=2097 Ack=1262
15	0.059	.80.69	.156.41	SSHv2	82	Client: New Keys
16	0.136	.156.41	.80.69	TCP	66	22 → 51813 [ACK] Seq=1262 Ack=2113
17	0.136	.80.69	.156.41	SSHv2	130	Client: Encrypted packet (len=64)
18	0.144	.156.41	.80.69	SSHv2	130	Server: Encrypted packet (len=64)
19	0.144	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=2177 Ack=1326
20	0.144	.80.69	.156.41	SSHv2	146	Client: Encrypted packet (len=80)
21	0.157	.156.41	.80.69	SSHv2	162	Server: Encrypted packet (len=96)
22	0.157	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=2257 Ack=1422
23	0.157	.80.69	.156.41	SSHv2	706	Client: Encrypted packet (len=640)
24	0.346	.156.41	.80.69	TCP	66	22 → 51813 [ACK] Seq=1422 Ack=2897
25	7.976	.156.41	.80.69	SSHv2	658	Server: Encrypted packet (len=592)
26	7.976	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=2897 Ack=2014
27	11.63	.80.69	.156.41	SSHv2	1234	Client: Encrypted packet (len=1168
28	11.64	.156.41	.80.69	SSHv2	114	Server: Encrypted packet (len=48)
29	11.64	.80.69	.156.41	TCP	66	51813 → 22 [ACK] Seq=4065 Ack=2062
30	11.64	.80.69	.156.41	SSHv2	146	Client: Encrypted packet (len=80)
31	11.76	.156.41	.80.69	TCP	66	22 → 51813 [ACK] Seq=2062 Ack=4145

Courtesy of WireShark

SSH Login

The **ssh** command has the following login syntax:

```
ssh [user@]<host> [command]
```

The options are explained on the right.

Example:

```
$ ssh dbarton@wksp2.isab.lacounty.gov
dbarton@wksp2.isab.lacounty.gov's password:
Last login: Mon Jul 29 13:10:01 2019 from 10.54.80.69
[dbarton@wksp2 ~]$
```

Option	Description
[user]	**Optional.** The user as recognized on the remote system. If supplied, it must be separated by the host with an "@". If not supplied, the user ID of the local shell is sent.
<host>	**Required.**
[command]	**Optional.** If absent, a remote shell is executed (the most common case). Otherwise the command is executed remotely. Control is returned to local shell after the command completion.

SSH Exercise 1

1. SSH to your remote account.
2. Create a new file in your home directory.

```
echo "Hello world" > greeting.txt
```

3. List the files in your directory: `ls`
4. Display the new file: `cat greeting.txt`
5. Log out.
6. Display the remote file through SSH using `cat greeting.txt` as the remote command.

```
ssh userid@hostname cat greeting.txt
```

SCP – Secure CoPy

The secure copy, `scp`, syntax is similar to the local command line copy command `cp`.

```
scp <from file> <to>
```

The <from file> is some file, either local or remote. The <to> parameter is either a directory in which to place the file (with the same name as from), or a complete file name specification.

The difference is the convention for specifying a remote file.

```
[user@]<host>:[file]
```

The syntax is similar to the `ssh` argument. The difference is the [file] that is separated from the host by colon instead of a space.

Examples:

The following two commands do the same thing.

```
scp dbarton@wksp2.la.gov:~/greeting.txt greeting.txt
scp dbarton@wksp2.la.gov:greeting.txt .
```

This is a common form of the scp command. It copies `greeting.txt` from the `dbarton` home directory on `wksp2.la.gov` to the current directory with the same name.

Wildcards are supported. To upload all local files with a `.cer` extension from the current local directory to the `trustedCerts` directory on Debbie's remote account (which we assume exists):

```
scp *.cer dbarton@wksp2.la.gov:trustedCerts
```

Note that this would result in an error if

- There were no `.cer` files in her current directory
- If there were more than one `.cer` file, but the `trustedCerts` directory did not exist.

If there were one `.cer` file and `trustedCerts` did not exist, it would be interpreted as the new filename.

SSH Exercise 2 – SCP

1. Copy the remote `greeting.txt` file to your local directory.
2. Copy it again, but name it locally as `farewell.txt`.
3. Change `farewell.txt` to say "Take care, World."
4. Copy `greeting.txt` and `farewell.txt` up to your remote folder.

Notes:

- Did the remote copy in step 4 ask you whether you wanted to replace the original `greeting.txt`?
- In Step 4, how did you specify the remote HOME folder? Try the following:

```
scp *.txt wksp2.la.gov
scp *.txt wksp2.la.gov:
scp *.txt wksp2.la.gov:~
```

S/FTP – Secure FTP

- FTP (File Transfer Protocol) is a TCP based protocol for exchanging files between two machines.
  - does not encrypt traffic in transit
  - employs two connections
    - one for data (file content)
    - one for commands
- S/FTP is an FTP exchange through an SSH connection (an SSH tunnel).
  - A single connection.
  - Not to be confused with FTP/S, which is completely different. Uses SSL instead of SSH. Does not support all FTP verbs.

SSH Exercise 3 – SFTP

1. Copy `greeting.txt` to `howdy.txt`.
2. Send `howdy.txt` to your remote machine.

```
sftp dbarton@wksp2.la.com
```

3. Follow the exchange below. Red indicates client input.

```
$ sftp dbarton@wksp2.isab.lacounty.gov
dbarton@wksp2.isab.lacounty.gov's password:
Connected to dbarton@wksp2.isab.lacounty.gov.
sftp> ls
farewell.txt   greeting.txt
sftp> lls
farewell.txt greeting.txt howdy.txt
sftp> put howdy.txt
Uploading howdy.txt to /home/dbarton/howdy.txt
howdy.txt
100%   12     0.5KB/s   00:00
sftp> bye
$
```

Note: `ls` = list files; `lls` = local list files

SSH Certificates

- So far all scenarios have authenticated through **knowledge** of the user's remote account **password**.
- Certificate login authenticates based on **possession** of a **private key** corresponding to an installed public key.
- Client
  1. generates public/private key pair,
  2. keeps private key securely stashed,
  3. installs public key on remote server.
- Subsequent SSH connections should use certificate authentication.

Server Certificate

- Always presented to client at beginning of handshake.
- Client is prompted to accept the server key.
  - If key is accepted, it is stored in client's `known_hosts` file.
  - Subsequent handshakes for a key already in the `known_hosts` file will not prompt the user.
  - If the client is prompted for a server's key later, that means it must have changed.
    - Server is spoofed or otherwise compromised.
    - Admin changed the key without telling anyone (usually the case).

The `ssh-keygen` Command

- Client Configuration Directory
  - On Linux and macOS this defaults to `$HOME/.ssh`
  - On Windows the default is `%USERPROFILE%\.ssh`
  - The permissions on this directory must permit the client **exclusive** access. Many SSL commands will silently fail if this directory is granted access to others.
- Sample Contents
  - `id_rsa` – private key; should not be visible or accessible by anyone else.
  - `id_rsa.pub` – public key; distributed to other hosts.
  - `known_hosts` – list of remote SSH hosts that we trust.
- The `ssh-keygen` command creates the first two files.

```
(base) isabmbp1:~/.ssh$ ls -la
drwx------    5 pglezen    160 Jul 23 13:31 ./
drwxr-xr-x+ 104 pglezen   3328 Aug 16 13:46 ../
-rw-------    1 pglezen   3326 Apr 11  2016 id_rsa
-rw-r--r--    1 pglezen    751 Apr 11  2016 id_rsa.pub
-rw-r--r--    1 pglezen  13965 Aug  8 07:24 known_hosts
```

Key Generation Syntax

- The basic form of the command:

```
ssh-keygen -t rsa -b 4096
```

  - type is RSA
  - key length is 4096 bits
  - private and public keys will be generated into the configuration directory: named `rsa` and `rsa.pub` respectively.
  - use `-f` flag to override base filename.
  - use `-C` flag to identify the owner.

```
$ ssh-keygen -t rsa -b 4096 -f sample
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in sample.
Your public key has been saved in sample.pub.
The key fingerprint is:
SHA256:tEqbrvR5cbdM1tXNAQ4QfRd17XR+3A7f++h6PT1i4ak
pglezen@isabmbp1.local
The key's randomart image is:
+---[RSA 4096]----+
|         o+. ...*|
|           .o. o=|
|        .   ...**|
|       . .    . @|
|      . S    . =o|
|     . +. . +.. +|
|    . +  o =..o.o|
|   . o ..   o=.=o|
|    ..+.   E+++ =|
+----[SHA256]-----+
```

Key Password

The sample on the previous slide opted out of assigning a password. This is a classic security-versus-convenience trade off. You should at least be aware of these trade-offs.

	Password (no caching)	Cached Password	No Password
Convenience	Enter your password every time you reference your private key.	Supporting applications allow remote logins without a password prompt.	SSH commands are entered as if authentication is not even required.
Key Stolen	Can't be used without password.	Can't be used without password.	Completely compromised.
Someone else using your machine.	Cannot issue SSH commands on your behalf from command line.	They have all the SSH access you have through supporting applications.	They have all the SSH access that you have.

SSH Exercise 4 – Generate Your Key Pair

Caution: If you already have a key pair with the default name and location, this command will overwrite it. Use the -f option to generate a pair with a different base filename.

1. Run the command

   ```
   ssh-keygen -t rsa -b 4096 -C "name <email>"
   ```

   the -C option adds information about you into the public key.
2. Check your SSH configuration directory for the key pair.
3. Display your public and private key at your command line.

SSH Lab 5 - Deploy your Public Key

- You can deploy your public key to any remote account to which you have access.
- Of course, you should already have user ID/password access.
- Just upload your public key the remote server (recall **scp**).
- Create a .ssh directory.
- Append certificate to the end of `~/.ssh/authorized_keys`

```
cat mycert.pub >> ~/.ssh/authorized_keys
```

- Troubleshooting
  - Most common source of errors is lack of restrictive file permissions.
    - `.ssh` folder and contents should only be accessible by owner (locally and remote).
  - To show ssh debugging info:
    - add `-v` flag
    - for more, add two: `-v -v`
    - log on the server is often in `/var/log/messages`. (Access to contents restricted to root.)

OpenSSH vs SSH2

- OpenSSH is common on
  - Linux
  - macOS
  - AIX
- SSH2 is common on
  - Windows, especially with PuTTY
- To convert SSH2 → OpenSSH
  - Run ssh-keygen on OpenSSH machine

SSH2 Certificate begins like this:

```
---- BEGIN SSH2 PUBLIC KEY ----
Comment: "rsa-key-20180504"
AAAAB3NzaC1yc2EAAAABJQAAAQEA9L1PUN69Mh4MkgK42JavxWd6TfOSMqZO9OP/
77VksSRCdoZRei/mOjM7LO5DEVPBb1HkTsrV6ut6qG1i9VUQF5JW5cv7kvNdVyrL
f2fpiNm/ReZuBsUNbAMqmJCehgCfvFas3fk1UoRIIg1vMvQVBh2dEi/H8agw13Am
```

OpenSSH Certificate begins like this:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEA9L1PUN69Mh4MkgK42JavxWd6TfOSMqZO9OP/
77VksSRCdoZRei/mOjM7LO5DEVPBb1HkTsrV6ut6qG1i9VUQF5JW5cv7kvNdVyrL
f2fpiNm/ReZuBsUNbAMqmJCehgCfvFas3fk1UoRIIg1vMvQVBh2dEi
```

Run this on OpenSSH machine:

```
$ ssh-keygen -i -f ssh2cert.pub > opensshcert.pub
```

Areas for Further Investigation

- Learn how to configure SSHD
  - We focused on the client-side in this workshop.
  - The server can be configured enable/disable various features
    - shell logins
    - SFTP
- Deploy your public key to GitHub or AWS CodeCommit.
  - SSH is more firewall friendly than HTTPS.
- Learn about SSH tunnels
  - Tunnel insecure protocols through a secure SSH tunnel.
  - May raise a few eyebrows among your networking colleagues.