Practical session 2, Introduction to SSH and bash commands

What is the Secure Shell (SSH) protocol?

The Secure Shell (SSH) protocol is a method for securely sending commands to a computer over an unsecured network. SSH uses cryptography to authenticate and encrypt connections between devices. SSH also allows for tunneling, or port forwarding, which is when data packets are able to cross networks that they would not otherwise be able to cross. SSH is often used for controlling servers remotely, for managing infrastructure, and for transferring files.

The protocol was designed to replace older, insecure remote shell protocols such as Telnet, Rlogin, and FTP, which transmit information, especially passwords, in plaintext, making them susceptible to interception and disclosure.

Public key cryptography

SSH is "secure" because it incorporates encryption and authentication via a process called public key cryptography. Public key cryptography is a way to encrypt data, or sign data, with two different keys. One of the keys, the public key, is available for anyone to use. The other key, the private key, is kept secret by its owner. Because the two keys correspond to each other, establishing the key owner's identity requires possession of the private key that goes with the public key.

These "asymmetric" keys — so called because they have different values — also make it possible for the two sides of the connection to negotiate identical, shared symmetric keys for further encryption over the channel. Once this negotiation is complete, the two sides use the symmetric keys to encrypt the data they exchange.

There are 2 common SSH keys algorithms, RSA and Ed25519

To generate SSH keys for both RSA and Ed25519 algorithms, you can use the `ssh-keygen` command available on most Unix-like operating systems, including Linux and macOS. Here's how to generate keys using both algorithms:

RSA Key Generation

For RSA, a key size of 2048 bits is generally considered the minimum for security, but 4096 bits is recommended for better security.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This command will create a new RSA key pair with a size of 4096 bits. The `-C` flag adds a comment in the key file, usually your email, for identification.

Ed25519 Key Generation

Ed25519 is a public-key signature system that uses a variant of the Schnorr signature based on Twisted Edwards curves. For Ed25519, the key size is fixed at 256 bits.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

This command generates a new Ed25519 key pair. Ed25519 keys provide a great balance between security and performance.

Benefits of Ed25519 over RSA

- **Performance**: Ed25519 offers faster key generation, signing, and verification compared to RSA. This makes it particularly advantageous for systems where these operations are performed frequently.
- **Security**: Ed25519 is designed to provide a high level of security against several types of cryptographic attacks. Its design choices help mitigate risks associated with side-channel attacks and other vulnerabilities that can affect RSA and other algorithms.
- **Key Size**: Ed25519 keys are much shorter than RSA keys for a comparable level of security. An Ed25519 public key is only 256 bits long. This compact size results in smaller and faster transmissions of keys and signatures. In contrast, RSA keys recommended for secure applications are significantly larger (at least 2048 bits, with 3072 or 4096 bits preferred for higher security levels).
- **Better Defaults**: The algorithm's design includes secure defaults that avoid many of the pitfalls associated with the complexity and flexibility of RSA. RSA's security can vary significantly depending on key size, padding

schemes, and other factors, making it easier to misconfigure in an insecure manner.

Configure password authentication for SSH:

Edit ssh config and restart the sshd:

```
sudo vim /etc/ssh/sshd_config
```

Look for the line `#PasswordAuthentication yes` in the file. If it is commented out (prefixed with `#`) or set to `no`, change it to `PasswordAuthentication yes`. This line explicitly enables password authentication on the SSH server.

```
sudo systemctl restart sshd
```

Configure different SSH port

To configure SSH to use a port other than the default port 22, you will need to edit the SSH server configuration file (`sshd_config`) on the server you're connecting to. This change will affect anyone trying to connect to the server via SSH, so it's important to ensure that the new port does not conflict with other services and is allowed through any firewalls.

Step 1: Edit the SSH Server Configuration

1. **Open the SSH Configuration File**: On most Linux systems, the SSH server configuration file is located at `/etc/ssh/sshd_config`. You'll need to edit this file with root privileges. You can use any text editor; in this example, we'll use `nano`.

   ```
   sudo vim /etc/ssh/sshd_config
   ```

2. **Change the Port Number**: Look for the line that starts with `#Port 22`. You might find it commented out (indicated by the `#` at the beginning). Uncomment this line by removing the `#`, and change `22` to your desired port number, for example, `2222`.

   Before:

   ```
   #Port 22
   ```

After:

```
Port 2222
```

3. **Save and Close the File**: After making the change, save the file and exit the text editor. If you're using `nano`, you can do this by pressing `Ctrl + X`, then `Y` to confirm the changes, and `Enter` to close.

Step 2: Adjust the Firewall Settings

If your server is using a firewall, you'll need to allow traffic on the new SSH port. The commands to do this will depend on your firewall software. Here are examples for `UFW` (Uncomplicated Firewall) and `firewalld`, which are commonly used on Linux systems.

- **For UFW**:

```
sudo ufw allow 2222/tcp
```

- **For firewalld**:

```
sudo firewall-cmd --permanent --add-port=2222/tcp
sudo firewall-cmd --reload
```

Replace `2222` with your chosen port number.

Step 3: Restart the SSH Service

To apply the changes, you need to restart the SSH service. The command to do this can vary slightly depending on your system's init system (SystemD or SysVinit).

- **For SystemD**:

```
sudo systemctl restart sshd
```

- **For SysVinit**:

```
sudo service sshd restart
```

Step 4: Connect Using the New Port

After changing the port and restarting the SSH service, you can connect to the server using the new port by specifying the `-p` option with the SSH command.

```
ssh -p 2222 -i "key.pem" user@your_server_ip
```

Replace `2222` with your chosen port number and `user@your_server_ip` with your actual username and server IP.

Important Note: Changing the SSH port is a simple security measure to avoid automated attacks and scans targeting the default port. However, it should not be the only security measure in place. Always use strong passwords or key-based authentication, configure firewalls properly, and keep your system updated for enhanced security.

Use ssh for secure file transfer:

Basic SCP Syntax

The basic syntax to copy files from a local system to a remote system is:

```
scp [OPTIONS] source_file_path username@remote_host:destinati
```

```
scp [OPTIONS] username@remote_host:source_file_path destinati
```

Typical usage with ssh key:

```
scp -i /path/to/private/key example.txt user@192.168.1.10:~
```

Main linux directories

- **/** (Root): The base of the filesystem. Every file and directory starts from here.
- **/bin**: Contains essential user binary files (programs) that must be available in single-user mode and for all users, such as `ls`, `cp`, etc.
- **/boot**: Holds files required to start the boot process, including the Linux kernel itself and the bootloader (GRUB, for example).

- **/dev**: This directory contains device files. In Linux, hardware devices are accessed just like files, and /dev contains special files that represent devices.
- **/etc**: Home to configuration files required by all programs. This directory contains the configuration files for the system.

  `cat /etc/passwd`

  `cat /etc/shadow`
- **/home**: The personal directory for users. Each user has a directory within /home for their files, settings, etc.

  ### Key Points About `.bashrc`:

  - **User-Specific Configuration**: Each user on the system can have their own `.bashrc` file for customizing their Bash sessions. This allows different users to have different configurations based on their preferences or needs.
  - **Common Uses**:
    - **Setting Environment Variables**: You can set environment variables that should be available in your shell sessions, like `PATH`, `EDITOR`, and others.
    - **Creating Aliases**: Shortcuts for longer commands can be defined here, making them quicker and easier to type.
    - **Customizing the Bash Prompt**: The appearance of the command prompt can be changed by modifying the `PS1` variable.
    - **Loading Shell Functions**: Functions that perform specific tasks can be defined in the `.bashrc` file for reuse across your sessions.
- **/lib**: Contains shared library files that support the binaries located in /bin and /sbin. Libraries needed for essential system operation.
- **/media** and **/mnt**: These directories are used for mounting removable and temporary media, like USB drives and temporary filesystems, respectively.
- **/opt**: Intended for the installation of add-on application software packages. A place for software not included in the default installation.
- **/proc**: A virtual filesystem that provides a mechanism for the kernel to send information to processes. It contains runtime system information (e.g., system memory, devices mounted, hardware configuration, etc.).

- **/root**: The home directory for the root user, separate from /home for security and organizational purposes.
- **/sbin**: Like /bin, this directory holds binary (executable) files, but for system administration purposes. These are generally not intended for direct execution by regular users.
- **/tmp**: Temporary files are placed here by system and applications. These may be deleted without warning upon reboot.
- **/usr**: Contains the majority of user utilities and applications, including system binaries, libraries for system binaries, program files, and more. It's a secondary hierarchy for read-only user data.
- **/var**: Variable files—files whose content is expected to continually change during normal operation of the system—such as logs, spool files, and temporary e-mail files.

Linux CLI

In Linux, the manual pages (or "man pages") provide detailed documentation about commands, system calls, configuration files, and more. They are an invaluable resource when working on the command line, offering syntax usage, options, and examples. To access the manual pages, you use the `man` command followed by the name of the command you want to learn about.

Basic Usage

To open the manual page for a specific command, type `man` followed by the command name. For example, to see the manual page for the `ls` command, you would use:

```
man ls
```

Navigating Manual Pages

Once you open a manual page, you can navigate using the following keys:

- **Down Arrow or `j`**: Move down one line.
- **Up Arrow or `k`**: Move up one line.
- **Space**: Move down one page.
- `b`: Move up one page.

- `/`: Search for a term. After pressing `/`, type your search query and press Enter. For example, `/option` will search for the term "option".
- `n`: After performing a search, press `n` to move to the next occurrence of the search term.
- `N`: Move to the previous occurrence of the search term.
- `q`: Quit the manual page and return to the command line.

Finding Commands

If you're unsure of the exact command name, you can use the `apropos` command to search the manual page descriptions for keywords. For example, to find commands related to "copy", you could use:

`apropos copy`

Basic commands required for home task:

- File Operations: `cp`, `mv`, `rm`, `touch`
- Navigation: `cd`, `ls`, `pwd`
- Directory Operations: `mkdir`, `rmdir`
- Text Manipulation: `cat`, `grep`, `sed`, `vim, nano`
- System Information: `uname`, `df`, `ps`

Typical use cases for uname:

The `uname` command in Unix and Linux operating systems is used to display system information. When run without any options, `uname` shows the operating system name.

- To see the kernel name:

```
bashCopy code
uname -s
```

- To get the kernel release:

```
bashCopy code
uname -r
```

- To display all available system information:

```
bashCopy code
uname -a
```

- User Management: `adduser`, `passwd`, `su`, `chown, chmod`

What is Bandit tool:

`https://overthewire.org/wargames/bandit/bandit0.html`

Hometask:

Solve Bandit levels 1-20, send me a list of keys in txt file.