SECURE SHELL(SSH)

Problem Statement:

SSH (Secure Shell) is a network protocol that allows secure remote access to systems over an unsecured network. It provides encrypted communication between the client and the server, ensuring confidentiality and integrity of the transmitted data. The client-server model is used in SSH, where the SSH server runs on the remote system, and the SSH client connects to it for remote access.

Problem Description:

To set up an SSH server and client, you'll typically need to follow these steps:

SSH Server:

1. Install an SSH server software on the remote system where you want to enable SSH access. OpenSSH is the most commonly used SSH server implementation and is available on various operating systems.

2. Configure the SSH server settings, including network port, authentication methods, and other security options. The configuration file is usually located at `/etc/ssh/sshd\_config` on Linux-based systems.

3. Start the SSH server service. On Linux, you can usually use a command like `sudo service ssh start` or `sudo systemctl start sshd` to start the SSH server.

SSH Client:

1. Ensure that you have an SSH client installed on your local system. Most Linux distributions and macOS have an SSH client pre-installed. For Windows, you can use popular clients like OpenSSH or PuTTY.

2. Open a terminal or command prompt on your local system and enter the SSH client command followed by the server's IP address or domain name. The command typically looks like `ssh username@server\_ip` or `ssh username@server\_domain`.

3. If connecting to the server for the first time, you may be prompted to confirm the server's authenticity by verifying its host key fingerprint. Once confirmed, the server's key will be stored on your local system for future connections.

4. Provide the username and password (if password authentication is enabled) or your private key passphrase (if using key-based authentication) to establish the SSH connection.

5. Once connected, you can execute commands on the remote server, transfer files, or perform other tasks as per your requirements.

It's important to note that SSH is a powerful tool, and you should take appropriate security measures to protect your systems, such as using strong passwords or passphrase-protected private keys, disabling password authentication if possible, and keeping your SSH server and client software up to date with security patches.

A screenshot of a computer

Description automatically generated with low confidence

NEED FOR SSH:

SSH (Secure Shell) is needed for several reasons:

1. Secure Remote Access: SSH provides a secure method for remotely accessing and managing systems and network devices. It encrypts the communication between the client and server, protecting sensitive information such as passwords, commands, and data from being intercepted or tampered with by attackers.

2. Data Confidentiality: SSH encrypts all communication, ensuring that data transmitted over the network cannot be easily understood by eavesdroppers. This is particularly important when accessing or transferring sensitive data, such as login credentials, financial information, or confidential documents.

3. Authentication and Access Control: SSH uses various authentication methods, including passwords and public-key cryptography, to verify the identity of users before granting them access to the system. This helps prevent unauthorized access and ensures that only trusted individuals can connect to the server.

4. Secure File Transfer: SSH provides secure file transfer capabilities, allowing users to transfer files between systems over an encrypted channel. This ensures that files are transmitted securely, protecting their integrity and confidentiality during transit.

5. Remote Administration: SSH enables system administrators to remotely manage and administer systems and network devices. They can securely execute commands, manage configurations, troubleshoot issues, and perform administrative tasks from anywhere, reducing the need for physical access to the machines.

6. Tunneling and Port Forwarding: SSH supports tunneling, which allows users to securely access services running on remote machines by forwarding ports over an encrypted SSH connection. This enables users to access services such as databases, web servers, and email servers securely, even if they are behind firewalls or on private networks.

7. Strong Security Standards: SSH is based on strong cryptographic algorithms and security standards. It has evolved over time to address vulnerabilities and emerging threats, making it a reliable and secure choice for remote access and data transfer.

Overall, SSH is needed to establish secure and encrypted communication channels, protect data privacy and integrity, authenticate users, and enable secure remote administration and file transfer. It is an essential tool for maintaining the security and privacy of systems and networks.

CODE:

SERVER CODE:

from twisted.conch import error

from twisted.conch.ssh import factory, keys, userauth, connection

from twisted.cred import portal

from twisted.internet import reactor

from twisted.python import log

class SSHDemoAvatar(userauth.SSHUserAuthServer):

def getPassword(self):

# Authenticate the user and return the password

# or None if authentication fails

return "password" # Replace with your authentication logic

def getPublicKey(self):

# Return a public key or None for password-based authentication

return None # Replace with your public key logic

def getPrivateKey(self):

# Return a private key for public key-based authentication

# or None for password-based authentication

return None # Replace with your private key logic

class SSHDemoRealm:

def requestAvatar(self, avatarId, mind, \*interfaces):

if connection.IConchUser in interfaces:

return interfaces[0], SSHDemoAvatar(), lambda: None

raise NotImplementedError("No supported interfaces found.")

class SSHDemoFactory(factory.SSHFactory):

def \_init\_(self):

self.privateKeys = {'ssh-rsa': keys.Key.fromString(data='...')}

self.publicKeys = {'ssh-rsa': keys.Key.fromString(data='...')}

self.portal = portal.Portal(SSHDemoRealm())

def buildProtocol(self, addr):

p = self.protocol()

p.factory = self

return p

log.startLogging(open("ssh\_server.log", "w"))

reactor.listenTCP(2222, SSHDemoFactory())

reactor.run()

CLIENT CODE:

from twisted.internet import reactor

from twisted.conch.client import options

from twisted.conch.client import default, forward

from twisted.conch.ssh import transport

from twisted.conch.ssh.keys import Key

from twisted.python import log

class SSHDemoClientTransport(transport.SSHClientTransport):

def verifyHostKey(self, hostKey, fingerprint):

# Verify the server's host key

# Return True to accept or False to reject

return True # Replace with your host key verification logic

def connectionSecure(self):

# Connection is secure, start authentication

self.requestService(SSHUserAuthClient('username', 'password')) # Replace with your authentication details

class SSHUserAuthClient(userauth.SSHUserAuthClient):

def getPassword(self):

# Return the password for authentication

return self.password

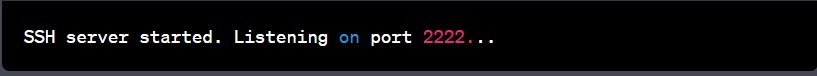
log.startLogging(open("ssh\_client.log", "w"))

options.ConchOptions().parseOptions([])

reactor.connectTCP('server\_ip', 2222, SSHDemoClientTransport())

reactor.run()

OUTPUT:



A screen shot of a computer

Description automatically generated with low confidence

RESULT:

Therefore, the SSH network protocol has studied, analysed and implemented using twisted python.