Python Networking with Telnet & SSH

Telnet(Terminal Network/ Teletype Network)

- Telnet is an interactive protocol that provides a virtual terminal for interaction.
- Telnet is a user command and an underlying TCP/IP protocol for accessing remote computers.
- Telnet is a type of network protocol which allows a user in one computer to logon to another computer which also belongs to the same network with whatever privileges user have been granted to the specific application and data on that computer.
- Telnet is an application layer protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection

Telnet server installation

- **To install telnet server**

  sudo apt-get install telnetd –y
- **To check the status of telnet service**

  sudo systemctl status inetd
- **Telnet to server from remote host**

  telnet server-ip-address
  Eg. telnet 192.168.1.5
- **Use telnet to Test Open Ports**

  telnet 192.168.0.100 80

Telnetlib Module

- The telnetlib module of python provides a Telnet class that implements the Telnet protocol.

  class telnetlib.Telnet(*host=None*, *port=0*[, *timeout*])
- Telnet represents a connection to a Telnet server.

✔ The host name and optional port number can be passed to the constructor

✔ The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).

Telnet Instance Methods

- **Telnet.read_until(*expected*, *timeout=None*)**

  Read until a given byte string, *expected*, is encountered or until *timeout* seconds have passed. When no match is found, return whatever is available instead, possibly empty bytes. Raise EOFError if the connection is closed and no cooked data is available
- **Telnet.read_all()**

  Read all data until EOF as bytes; block until connection closed.
- **Telnet.read_some()**

  Read at least one byte of cooked data unless EOF is hit. Return b'' if EOF is hit. Block if no data is immediately available.
- **Telnet.open(*host*, *port=0*[, *timeout*])**

  Connect to a host. The optional second argument is the port number, which defaults to the standard Telnet port (23). The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).

Telnet Instance Methods

- **Telnet.open(*host*, *port=0*[, *timeout*])**

  Connect to a host. The optional second argument is the port number, which defaults to the standard Telnet port (23). The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).
- **Telnet.close()**

  Close the connection
- **Telnet.get_socket()**

  Return the socket object used internally
- **Telnet.fileno()**

  Return the file descriptor of the socket object used internally
- **Telnet.write(*buffer*)**

  Write a byte string to the socket, doubling any IAC characters. This can block if the connection is blocked. May raise socket.error if the connection is closed.

Telnet using telnetlib

```
import getpass
import telnetlib

HOST = "localhost"
user = input("Enter your remote account: ")
password = getpass.getpass()

tn = telnetlib.Telnet(HOST)

tn.read_until(b"login: ")
tn.write(user.encode('ascii') + b"\n")
if password:
    tn.read_until(b"Password: ")
    tn.write(password.encode('ascii') + b"\n")

tn.write(b"ls\n")
tn.write(b"exit\n")

print(tn.read_all().decode('ascii'))
```

SSH Protocol

- SSH stands for Secure Shell, and is a program for logging into a remote machine and for executing commands on a remote machine.
- It was designed to be a secure alternative to previous access methods such as telnet.
- SSH encrypts all the data that is sent and received from the remote server, and offers secure authentication using SSH keys.

SSH key based authentication

- For key based authentication SSH uses SSH key pairs- a private key and a public key.
- The private key should always stay on local computer, and should not to lose it or let it fall into malicious hands. The public part of the key that should be uploaded to a remote machine to be accessed via SSH.
- By combining these two keys, a remote machine can determine its identity. When connecting to a remote machine with which the public key is already shared, it will send a challenge-type response and ask the local machine to identify with its private key.
- SSH keys should be manually generated using commands. The public keys should be placed inside the **~/.ssh/authorized_keys** in the home directory of the user you want to use on the remote machine.

SSH using Python

SSH can be implemented in Python using **paramiko module, pexpect, fabric and subprocess module.**

- Python's paramiko module gives an abstraction of the SSH protocol with both the client side and server side functionality.
- An instance of the Paramiko.SSHClient can be used to make connections to the remote server and transfer files

Important functions of paramiko API

- **paramiko.client.AutoAddPolicy:** Policy for automatically adding the hostname and new host key to the local HostKeys object, and saving it. This is used by SSHClient.
- **paramiko.client.MissingHostKeyPolicy:** Interface for defining the policy that SSHClient should use when the SSH server's hostname is not in either the system host keys or the application's keys. Pre-made classes implement policies for automatically adding the key to the application's HostKeys object (AutoAddPolicy), and for automatically rejecting the key (RejectPolicy).
- **paramiko.client.SSHClient:** A high-level representation of a session with an SSH server. This class wraps Transport, Channel, and SFTPClient to take care of most aspects of authenticating and opening channels.
- **connect(*hostname*, *username=None*, *password=None):*** Connect to an SSH server and authenticate to it. The server's host key is checked against the system host keys (see and any local host keys. If the server's hostname is not found in either set of host keys, the missing host key policy is used. The default policy is to reject the key and raise an **SSHException**.
- **exec_command(*command*):** Execute a command on the SSH server. A new **Channel** is opened and the requested command is executed. The command's input and output streams are returned as Python file-like objects representing stdin, stdout, and stderr

Paramiko Module(Used to work with remote server)

- Paramiko module will create a SSH Client and by using this it will connect to remote server and executes commands.
- We can transfer files from the remote machine to the local or vice versa using paramiko.
- Two ways to connect with remote server

 Using username and password

 Using username and cryptographic key

Installation of Paramiko

- ✔ sudo apt-get update(to update the system)
- ✔ sudo apt install python3-pip(pip3 installation for python3 packages)
- ✔ pip3 --version(to check the version)
- ✔ python3.5 -m pip install --upgrade pip(upgrading the pip)
- ✔ pip3 install paramiko(paramiko installation for python 3)
- ✔ python3 filename.py(Executing python file)

SSH Server Installation

- ✔ sudo apt-get install ssh
- ✔ sudo service ssh status
- ✔ sudo service ssh start
- ✔ /etc/ssh/sshd_config

Authentication using Username & Password

```
import paramiko
ssh_client=paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname='172.17.17.221',username='kjsce',
password='admin', port=22)
stdin, stdout, stderr=ssh_client.exec_command(‘ls’)
print('the output is')
print(stdout.read())
```

Authentication using Username & Key

```
import paramiko
ssh_client=paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname='172.17.17.221',username='kjsce', key_filename=‘private key file name with complete path’, port=22)
stdin, stdout, stderr=ssh_client.exec_command(‘ls’)
print('the output is')
print(stdout.read())
```

File Transfer using SFTP

```
import paramiko
ssh_client=paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname='172.17.17.221',username='kjsce', password='admin', port=22)
sftp_client=ssh_client.open_sftp()
sftp_client.get('remote server file name with path','downloaded_file.txt')
sftp_client.close()
ssh_client.close()
```