Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.Dibujo en blanco y negro

El contenido generado por IA puede ser incorrecto.

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Aplicaciones para comunicaciones en red

Academia “Sistemas Distribuidos”

Plan 2020

Práctica 17:

“**Telnet en Python**”

2015090269

González González Armando Omar

Profesor: Ojeda Santillan Rodrigo

Contenido

[Objetivo 3](#_Toc201741121)

[Introducción 3](#_Toc201741122)

[Desarrollo 4](#_Toc201741123)

[3.1 Implementación del Servidor Telnet (server.py) 4](#_Toc201741124)

[3.2 Implementación del Cliente Telnet (client.py) 5](#_Toc201741125)

[Conclusión 7](#_Toc201741126)

[Pregunta 7](#_Toc201741127)

[¿Cómo podrían generar un negocio a través de lo visto en la práctica? 7](#_Toc201741128)

[Bibliografía 7](#_Toc201741129)

## Objetivo

Desarrollar una aplicación cliente-servidor en Python que simule el protocolo Telnet para realizar un proceso de autenticación de usuario. El objetivo principal es comprender los fundamentos de la programación de sockets en Python, estableciendo una conexión de red para el intercambio de mensajes de texto plano entre un cliente y un servidor, y realizando una validación de credenciales.

## Introducción

Telnet es un protocolo de red que permite a un usuario acceder y gestionar un sistema remoto a través de una interfaz de línea de comandos basada en texto. Aunque en la actualidad ha sido en gran parte suplantado por protocolos más seguros como SSH, Telnet sigue siendo una herramienta educativa muy valiosa para aprender los conceptos básicos de la comunicación en red y la administración de sistemas remotos.

Para implementar este tipo de comunicación en Python, se utiliza la biblioteca socket, que proporciona una interfaz de bajo nivel para la programación de redes. Esta biblioteca permite la creación de clientes y servidores que se comunican mediante el intercambio de mensajes a través de sockets de red. En el contexto de esta práctica, socket es fundamental para establecer un servidor Telnet que escuche conexiones, y un cliente que se conecte a él para llevar a cabo un proceso de autenticación simple, demostrando el flujo de datos en texto claro característico del protocolo.

## Desarrollo

Para esta práctica, se crearon dos scripts en Python: un servidor que escucha conexiones y valida credenciales, y un cliente que se conecta al servidor para autenticarse.

### **3.1 Implementación del Servidor Telnet (**server.py**)**

El script del servidor tiene la responsabilidad de esperar conexiones entrantes, solicitar credenciales al cliente y validarlas. Su lógica de operación es la siguiente:

1. **Creación y Configuración del Socket:** Se crea un socket de tipo AF\_INET (IPv4) y SOCK\_STREAM (TCP). Se enlaza (bind) a la dirección 0.0.0.0 para escuchar en todas las interfaces de red disponibles y a un puerto específico (5555). Finalmente, se pone en modo de escucha con listen().
2. **Aceptación de Conexiones:** El servidor entra en un bucle infinito (while True) para aceptar continuamente conexiones de clientes mediante server\_socket.accept().
3. **Proceso de Autenticación:** Por cada cliente conectado, el servidor envía un mensaje de bienvenida y solicita el Login:. Recibe el nombre de usuario, solicita la Password: y recibe la contraseña.
4. **Validación:** Compara las credenciales recibidas con los valores predefinidos (username == 'escom' y password == 'cisco').
5. **Respuesta y Cierre:** Envía un mensaje de "Autenticación exitosa" o "Autenticación fallida" según el resultado de la validación y cierra la conexión con ese cliente, quedando a la espera de una nueva conexión.

import socket

# Configuración del servidor

HOST = '0.0.0.0' # Escuchar en todas las interfaces de red disponibles

PORT = 5555 # Puerto arbitrario que no requiere permisos de superusuario

# Crear el socket del servidor

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Servidor Telnet escuchando en {HOST}:{PORT}...")

while True:

client\_socket, client\_address = server\_socket.accept()

print(f"Conexión desde {client\_address}")

client\_socket.sendall("Bienvenido al servidor Telnet!\nLogin: ".encode('utf-8'))

# Recibir nombre de usuario

username = client\_socket.recv(1024).strip().decode('utf-8')

# Pedir contraseña

client\_socket.sendall("Password: ".encode('utf-8'))

password = client\_socket.recv(1024).strip().decode('utf-8')

# Autenticación simple

if username == 'escom' and password == 'cisco':

client\_socket.sendall("\nAutenticación exitosa!\n".encode('utf-8'))

else:

client\_socket.sendall("\nAutenticación fallida!\n".encode('utf-8'))

client\_socket.close()

### **3.2 Implementación del Cliente Telnet (**client.py**)**

El script del cliente se encarga de iniciar la conexión con el servidor y de interactuar con el usuario para enviar las credenciales.

1. **Creación y Conexión del Socket:** Se crea un socket TCP/IP y se utiliza el método connect() para establecer una conexión con la dirección IP y el puerto del servidor.
2. **Interacción con el Servidor:**
   * Recibe e imprime el mensaje de bienvenida y la solicitud de Login:.
   * Pide al usuario que ingrese su nombre de usuario por teclado y lo envía al servidor.
   * Recibe e imprime la solicitud de Password:.
   * Pide al usuario que ingrese su contraseña y la envía al servidor.
3. **Resultado y Cierre:** Recibe e imprime el mensaje final de autenticación (exitosa o fallida) y cierra la conexión.

import socket

# Configuración del cliente

HOST = '127.0.0.1' # Reemplaza con la IP de tu servidor Ubuntu

PORT = 5555 # Debe coincidir con el puerto configurado en el servidor

# Conectar al servidor

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

# Recibir el mensaje de bienvenida

print(client\_socket.recv(1024).decode('utf-8'))

# Enviar nombre de usuario

username = input("Ingrese su nombre de usuario: ")

client\_socket.sendall(username.encode('utf-8') + b"\n")

# Recibir la solicitud de contraseña

print(client\_socket.recv(1024).decode('utf-8'))

# Enviar la contraseña

password = input("Ingrese su contraseña: ")

client\_socket.sendall(password.encode('utf-8') + b"\n")

# Recibir el resultado de la autenticación

print(client\_socket.recv(1024).decode('utf-8'))

client\_socket.close()

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 1. Ejecución del servidor telnet.

A screenshot of a computer program

AI-generated content may be incorrect.  
Figura 2. Ejecución del cliente con autenticación fallida y exitosa.

## Conclusión

La práctica se completó de manera exitosa, logrando implementar un sistema de autenticación funcional que simula el protocolo Telnet mediante el uso de la biblioteca socket de Python. Se comprendió el flujo de comunicación básico en un modelo cliente-servidor: la creación, enlace y escucha por parte del servidor, y la conexión, envío y recepción de datos por parte del cliente.

Se concluye que la programación de sockets, aunque de bajo nivel, es fundamental para entender cómo operan los protocolos de red. La práctica también evidenció la principal debilidad de Telnet: toda la comunicación, incluidas las credenciales de usuario y contraseña, se transmite en texto plano, lo que lo hace inseguro para su uso en redes no confiables.

## Pregunta

### ¿Cómo podrían generar un negocio a través de lo visto en la práctica?

Aunque Telnet es obsoleto, el principio de administración remota es vital. Un negocio moderno podría ofrecer servicios de gestión y monitoreo automatizado de dispositivos de red (routers, switches, firewalls) para empresas. Utilizando protocolos seguros como SSH, se podría desarrollar una plataforma que permita a los administradores configurar y auditar cientos de dispositivos de forma centralizada, ofreciendo este servicio bajo un modelo de suscripción (SaaS).

## Bibliografía

[1] J. Postel and J. Reynolds, "RFC 854: Telnet Protocol Specification," Internet Engineering Task Force (IETF), May 1983. [En línea]. Disponible: <https://www.rfc-editor.org/rfc/rfc854>.

[2] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 8th ed. Boston: Pearson, 2021.

[3] The Python Software Foundation, "socket — Low-level networking interface," Python 3.12.4 documentation. [En línea]. Disponible: <https://docs.python.org/3/library/socket.html>. [Consultado: 25-jun-2025]