

INSTITUTO TECNOLÓGICO DE SAN JUAN DEL RÍO



Tópicos de ciberseguridad.

Práctica - Parchar un ejecutable.

P R E S E N T A:

CRISTIAN MARTINEZ MARTINEZ 21590392

JESÚS CRUZ CRUZ 21590382

SALVADOR RAMIREZ HERNANDEZ 20590284

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PERIODO AGOSTO-DICIEMBRE 2025

Descripción: A partir de un ejecutable con un validación en línea proporcionado por el docente, aplicar los conocimientos previos adquiridos para saltar la validación.

R005:

El objetivo del procedimiento fue interceptar y controlar la comunicación de la aplicación phackeame.exe con su servidor remoto para pruebas y análisis. Para lograrlo, se siguieron tres pasos principales:

1. **Identificación del destino de la comunicación:** Se utilizó tshark para capturar las solicitudes HTTP/HTTPS generadas por el ejecutable y así determinar la IP y puerto del servidor al que se conectaba.
2. **Creación de un servidor local simulado:** Se implementó un servicio Python usando Flask que escucha en 127.0.0.1:8080 y devuelve siempre la misma respuesta JSON {"R":200,"D":50} a las solicitudes del ejecutable, sin reenviar peticiones a internet.
3. **Parcheo del ejecutable:** Se modificó la IP del servidor dentro del binario (phackeame.exe) para redirigir las peticiones hacia el servidor local, garantizando que el ejecutable consumiera el mock server en lugar del servidor real.

Funcionamiento:

Cuando se ejecuta el programa parcheado, este envía sus solicitudes al servidor local implementado en Python, que responde automáticamente con los datos simulados. Esto permite controlar y verificar la respuesta de la aplicación sin depender de la infraestructura original del profesor.

Propósito:

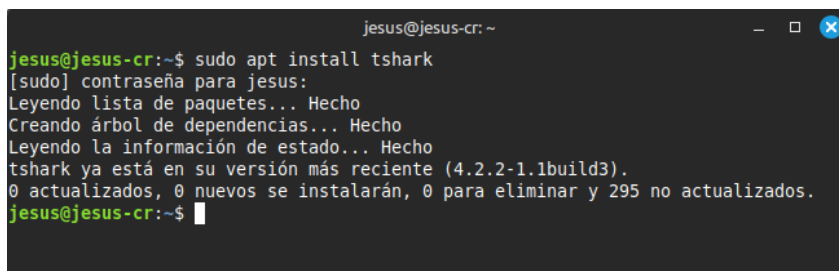
El procedimiento se realizó con fines de análisis y pruebas, evitando la comunicación con servidores externos y facilitando la experimentación con la aplicación de manera segura y reproducible.

Pasos:

1) Instalar y usar tshark para identificar host/IP destino

1.1 Instalación de tshark.

- Acción: Se instaló la herramienta de captura de paquetes en la máquina (Linux Mint) mediante el gestor de paquetes.
- En la **Imagen1** se captura la terminal mostrando el comando de instalación (sudo apt install tshark) y la salida del sistema indicando que la instalación se completó correctamente.

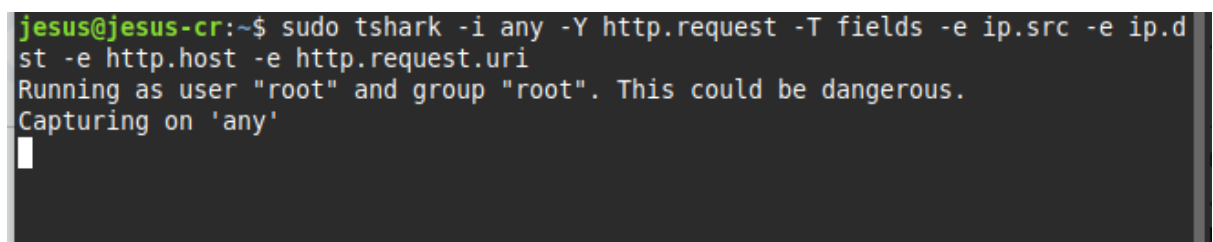


```
jesus@jesus-cr: ~  
jesus@jesus-cr:~$ sudo apt install tshark  
[sudo] contraseña para jesus:  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
tshark ya está en su versión más reciente (4.2.2-1.1build3).  
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 295 no actualizados.  
jesus@jesus-cr:~$
```

Imagen1. Se aprecia la ejecución del comando y los mensajes del gestor que confirman la descarga e instalación de tshark.

1.2 Preparar la captura de solicitudes HTTP/HTTPS.

- Acción: Se ejecutó tshark con el comando: **sudo tshark -i any -Y http.request -T fields -e ip.src -e ip.dst -e http.host -e http.request.uri** para listar en tiempo real solicitudes HTTP y campos clave (IP origen, IP destino, host y URI).
- En la **Imagen1.2** captura la terminal con el comando en ejecución y la salida en tiempo real.



```
jesus@jesus-cr:~$ sudo tshark -i any -Y http.request -T fields -e ip.src -e ip.d  
st -e http.host -e http.request.uri  
Running as user "root" and group "root". This could be dangerous.  
Capturing on 'any'  
[...]
```

Imagen1.2. La salida muestra líneas con ip.src, ip.dst, http.host y http.request.uri; esta información permite localizar a qué IP o dominio y qué ruta (ej. /login) el ejecutable envía peticiones.

1.3 Ejecutar y probar phackeame.exe.

- Acción: Se ejecutó phackeame.exe con Wine y se rellenaron los campos de acceso de prueba: Usuario: Magenta, Contraseña: 12345678.
- En la **Imagen1.3** se captura la interfaz del programa mostrando los campos con los valores indicados.

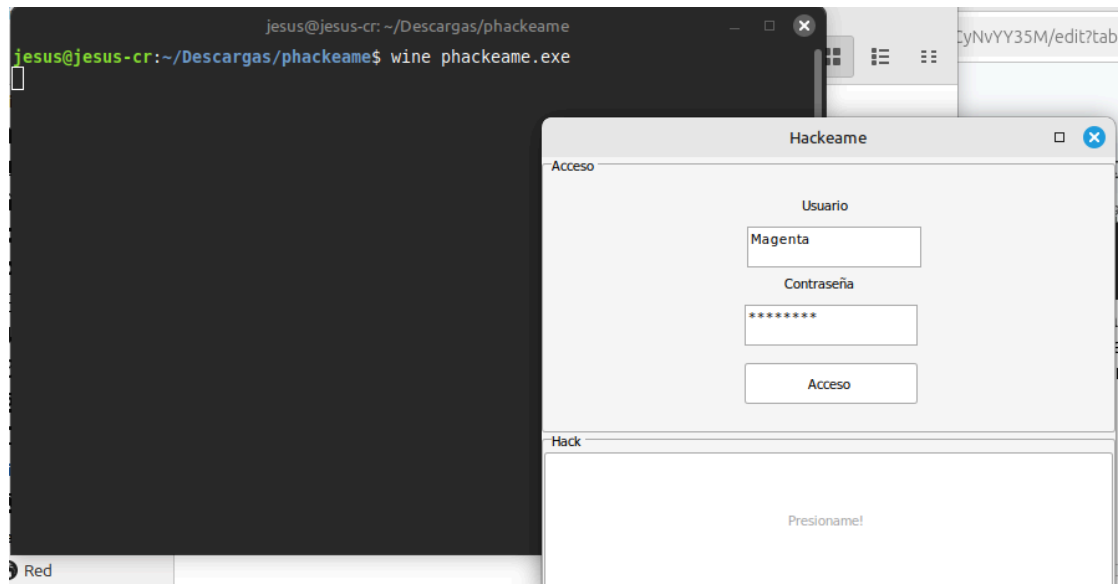


Imagen1.3. El formulario del ejecutable está listo para enviar la solicitud de autenticación.

1.4 Generación de la solicitud y evidencias en tshark.

- Acción: Al pulsar el botón “Acceso” en la aplicación, la petición apareció en la salida de tshark.
- En la **Imagen1.4** se captura tshark donde se ve la nueva línea correspondiente a la solicitud del exe (con IP destino).

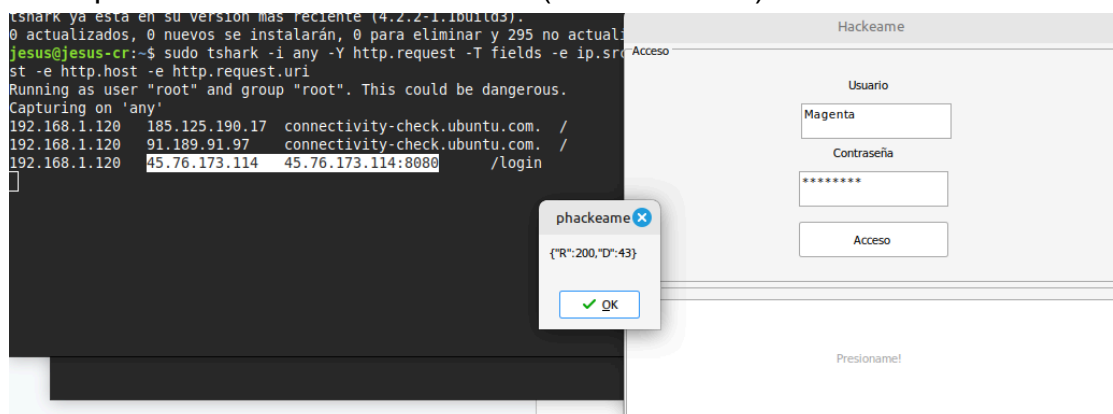


Imagen1.4 Evidencia de la comunicación saliente del exe hacia el servidor

remoto; el registro obtenido fue usado para identificar la IP/puerto objetivo que debía interceptarse.

2) Implementar el servicio Python.

- Acción: Se creó el archivo api.py con un servidor Flask que expone /login y devuelve siempre {"R":200,"D":50}.

```
1 from flask import Flask, request, Response
2 import json
3
4 app = Flask(__name__)
5
6 @app.route('/login', methods=['POST'])
7 def login():
8     # Ignorar lo que venga del exe, siempre devuelve D=50
9     payload = {"R": 200, "D": 50}
10    return Response(json.dumps(payload), status=200, mimetype='application/json')
11
12 if __name__ == "__main__":
13     app.run(host="0.0.0.0", port=8080)
14
```

Imagen2. Captura del editor mostrando proxy.py con el código anterior.

Nota: El archivo contiene la ruta /login que responde con JSON fijo y la configuración para que Flask escuche en el puerto 8080.

3) Instalar soporte para entornos Python (virtual env)

- Acción: Se instaló el paquete del sistema python3-venv y python3-pip: sudo apt install python3-venv python3-pip

Esto se hizo porque la instalación directa con pip estaba restringida por la política PEP 668 ("externally managed environment") en Linux Mint; por tanto se optó por crear un entorno virtual para instalar Flask y Requests sin afectar el Python del sistema.

- En la **Imagen3** se captura la terminal durante la instalación del paquete (sudo apt install python3-venv python3-pip).

```
jesus@jesus-cr:~/Descargas/phackeame$ sudo apt install python3-venv python3-pip
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
python3-venv ya está en su versión más reciente (3.12.3-0ubuntu2).
python3-pip ya está en su versión más reciente (24.0+dfsg-1ubuntu1.2).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 295 no actualizados.
jesus@jesus-cr:~/Descargas/phackeame$
```

Imagen3. Evidencia de la instalación de las herramientas necesarias para crear y administrar entornos virtuales.

4) Crear y activar el entorno virtual en la carpeta del proyecto

- Acción: En la carpeta del proyecto (/home/Usuario/Descargas/phackeame) se creó y activó un entorno virtual:

```
python3 -m venv venv
```

```
source venv/bin/activate
```

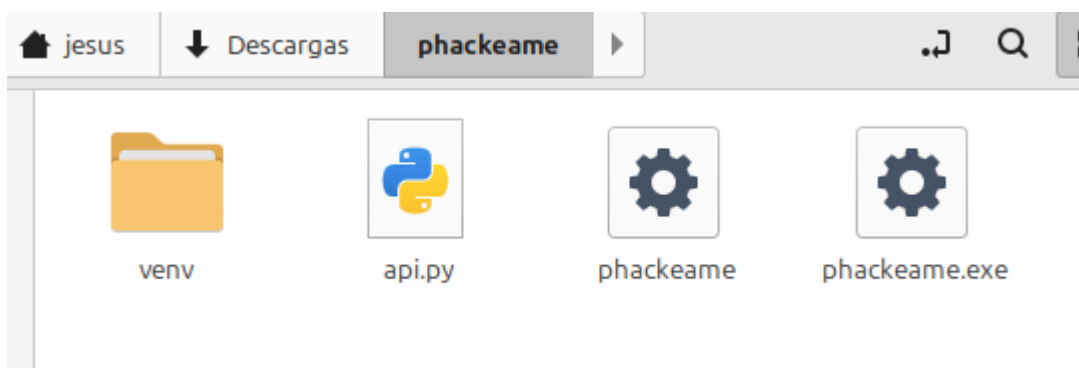


Imagen 4.1: Carpeta del entorno creada.

```
jesus@jesus-cr:~/Descargas/phackeame$ source venv/bin/activate  
(venv) jesus@jesus-cr:~/Descargas/phackeame$
```

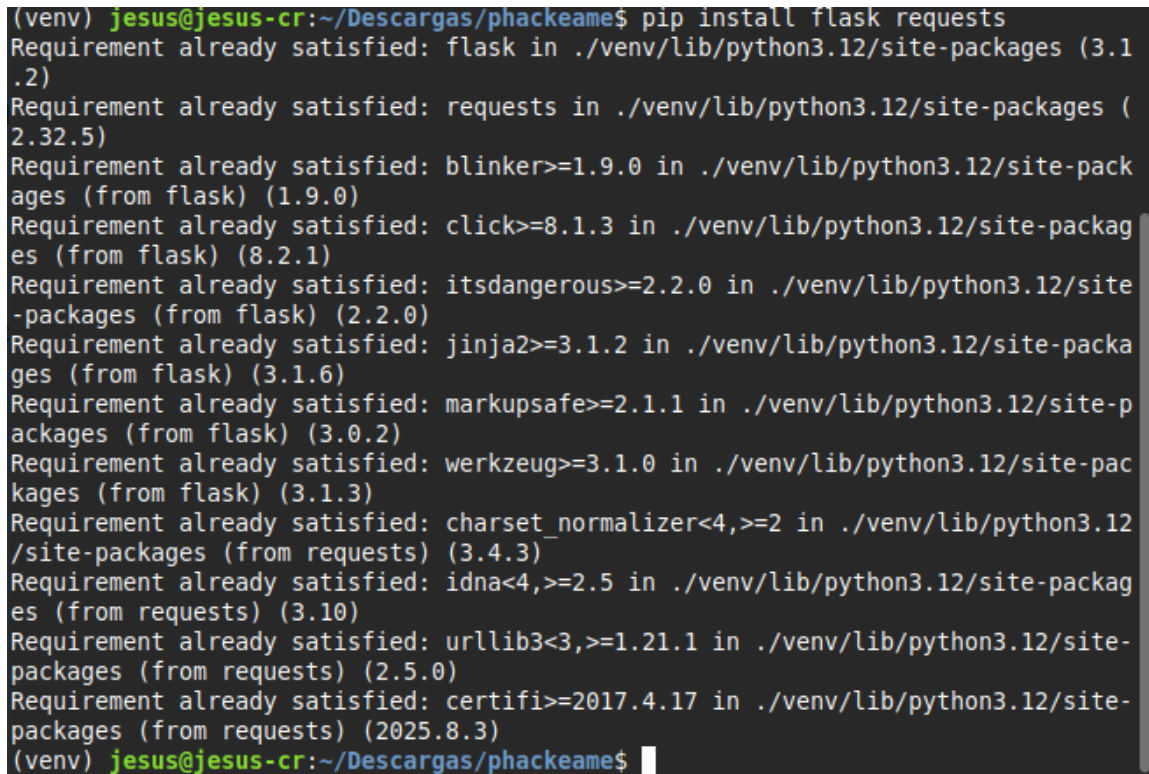
Imagen 4.2: terminal mostrando el prompt con (venv) luego de ejecutar source venv/bin/activate.

Se confirma que el entorno virtual está activo y se usará para instalar dependencias localmente.

5) Instalar Flask y Requests en el entorno virtual

- Acción: Con el entorno virtual activado se instalaron las dependencias:

pip install flask requests



```
(venv) jesus@jesus-cr:~/Descargas/phackeame$ pip install flask requests
Requirement already satisfied: flask in ./venv/lib/python3.12/site-packages (3.1.2)
Requirement already satisfied: requests in ./venv/lib/python3.12/site-packages (2.32.5)
Requirement already satisfied: blinker>=1.9.0 in ./venv/lib/python3.12/site-packages (from flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in ./venv/lib/python3.12/site-packages (from flask) (8.2.1)
Requirement already satisfied: itsdangerous>=2.2.0 in ./venv/lib/python3.12/site-packages (from flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in ./venv/lib/python3.12/site-packages (from flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in ./venv/lib/python3.12/site-packages (from flask) (3.0.2)
Requirement already satisfied: werkzeug>=3.1.0 in ./venv/lib/python3.12/site-packages (from flask) (3.1.3)
Requirement already satisfied: charset_normalizer<4,>=2 in ./venv/lib/python3.12/site-packages (from requests) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in ./venv/lib/python3.12/site-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./venv/lib/python3.12/site-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in ./venv/lib/python3.12/site-packages (from requests) (2025.8.3)
(venv) jesus@jesus-cr:~/Descargas/phackeame$
```

Imagen5: captura de la terminal con la ejecución del comando pip install flask requests y la salida que confirma la instalación.

6) Parchear el ejecutable (phackeame.exe) para apuntar a 127.0.0.1

6.1 Buscar IPs incrustadas en el binario.

- Acción: Se usó strings y grep para localizar direcciones IPv4 dentro del ejecutable: `strings phackeame.exe | grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}"`

```

jesus@jesus-cr:~/Descargas/phackeame$ strings phackeame.exe | grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}"
3.4.0.0
http://45.76.173.114:8080/login
TLazWriterTiff - Lazarus LCL: 3.4.0.0 - FPC: 3.2.2
TTiffImage - Lazarus LCL: 3.4.0.0 - FPC: 3.2.2
3.4.0.0
<assemblyIdentity version="1.0.0.0" processorArchitecture="*" name="CompanyName.ProductName.AppName" type="win32"/>
<assemblyIdentity type="win32" name="Microsoft.Windows.Common-Controls" version="6.0.0.0" processorArchitecture="*" publicKeyToken="6595b64144ccf1df" language="*" />
jesus@jesus-cr:~/Descargas/phackeame$

```

Imagen6.1: Captura de la terminal con el comando y la salida listando las IPs encontradas, incluida la IP del servidor del profesor.

Descripción de la imagen: lista de cadenas ASCII dentro del binario que facilitó ubicar la IP objetivo a reemplazar.

6.2 Abrir el binario en un editor hexadecimal.

- Acción: Se abrió phackeame.exe con ghex (u otro editor hex) para editar la sección donde aparecía la IP.

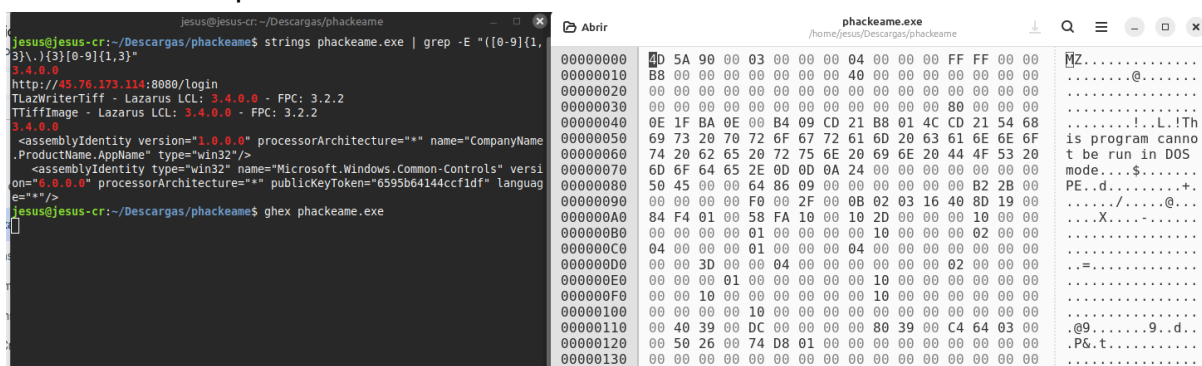


Imagen6.2: captura de la terminal con el comando que lanza ghex y la ventana del editor mostrando el contenido hex/ASCII del binario.

Descripción de la imagen: vista del binario en modo hexadecimal, lista para búsqueda y edición segura.

6.3 Localizar y reemplazar la IP dentro del binario.

- Acción: En el editor hex se localizó la secuencia ASCII de la IP del servidor (ej. 45.76.173.114) y se reemplazó por 127.0.0.1. Para mantener la integridad del binario, los bytes sobrantes se rellenaron con 00 00 00 00 (padding) de modo que la longitud del campo no cambiara.

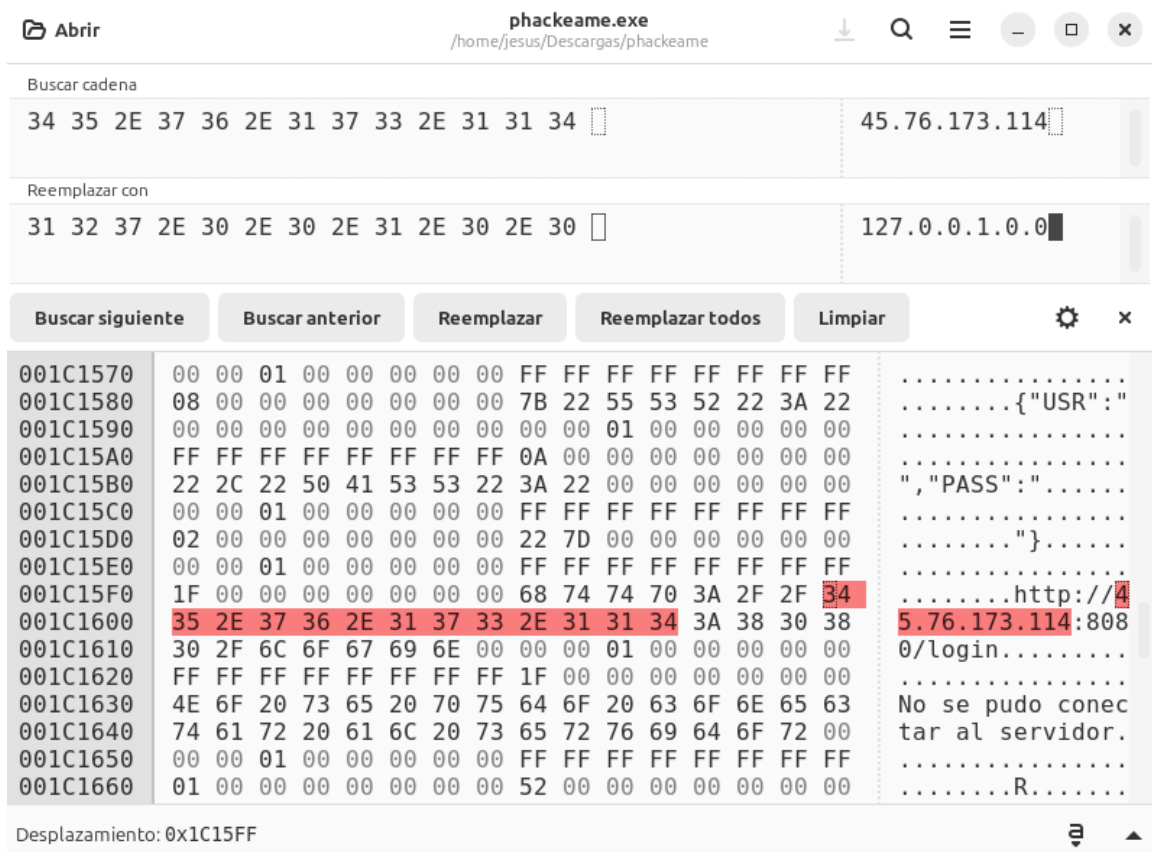


Imagen6.3: captura del editor hex mostrando la IP original resaltada y su reemplazo por 127.0.0.1 con bytes 00 añadidos como padding.

Descripción de la imagen: muestra el reemplazo seguro que fuerza al ejecutable a conectarse a localhost sin corromper offsets internos.

7) Ejecutar el mock server (api.py)

- Acción: Con el entorno virtual activo se inició el servidor Flask que responde en 127.0.0.1:8080:

```
jesus@jesus-cr:~/Descargas/phackeame$ ghex phackeame.exe
jesus@jesus-cr:~/Descargas/phackeame$ python3 api.py
* Serving Flask app 'api'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://192.168.1.120:8080
Press CTRL+C to quit
```

Imagen7. Captura de la terminal mostrando la salida de Flask indicando que el servidor está corriendo en http://127.0.0.1:8080.

Descripción de la imagen: el servicio local está activo y listo para recibir las solicitudes del ejecutable parcheado.

8) Probar el ejecutable parcheado

- Acción: Se ejecutó phackeame_patched.exe mediante Wine, se rellenaron los campos de prueba (Usuario: Magenta, Contraseña: 12345678) y se pulsó "Acceso". La petición fue dirigida a 127.0.0.1:8080 y el mock server respondió con:

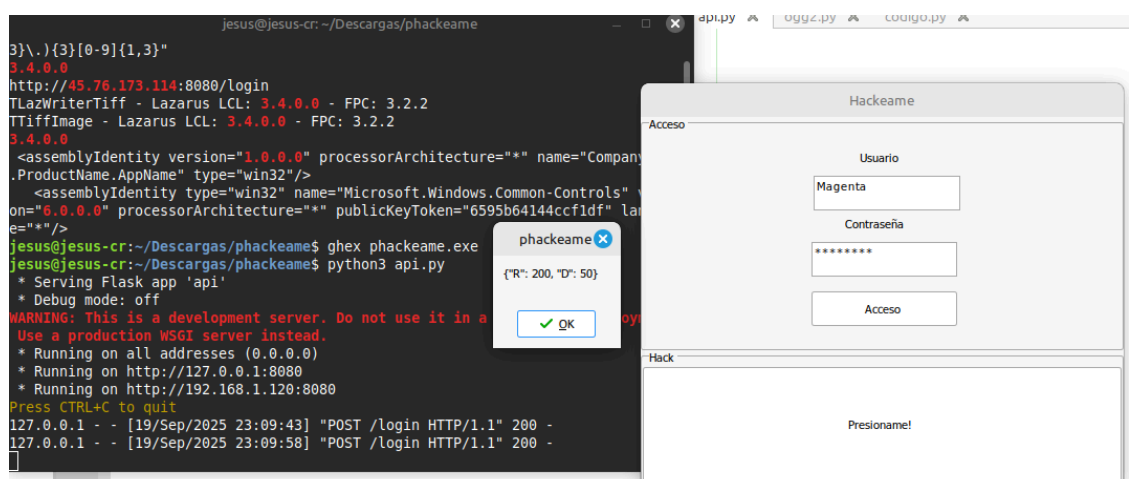


Imagen8: captura de la aplicación mostrando la respuesta JSON devuelta o captura del terminal del mock server mostrando la petición entrante y la respuesta enviada.

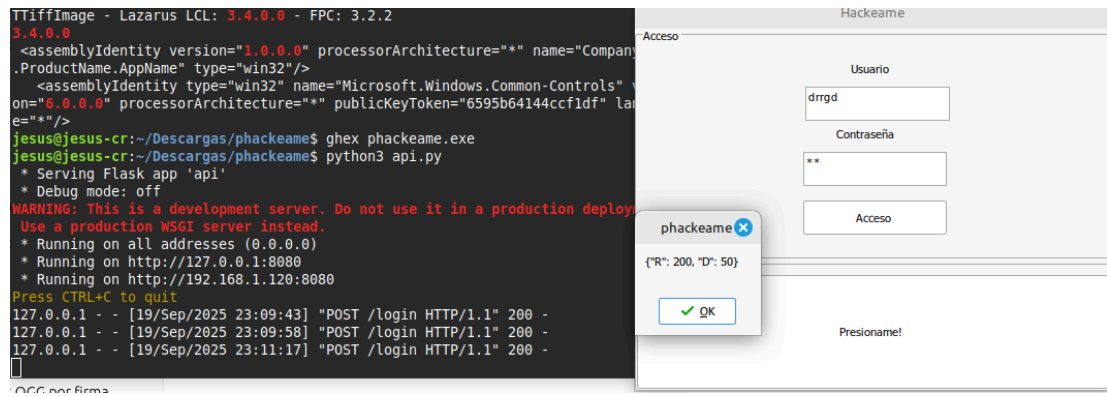


Imagen8.1: Ingreso con cualquier tipo de información.