Instituto Tecnológico de San Juan del Río

Tópicos de Ciberseguridad

Actividad 2

Parchado de .exe

P R E S E N T A N:

HERNÁNDEZ LUCIO ISAAC 21590386
MARTINEZ YAÑEZ ALEXIS JONATHAN 21590291
URIBE CALLEJAS JOSÉ URIEL 21021001

DOCENTE:

LEONARDO VALDES ARTEAGA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PERIODO [Enero - Junio 2026]

Índice

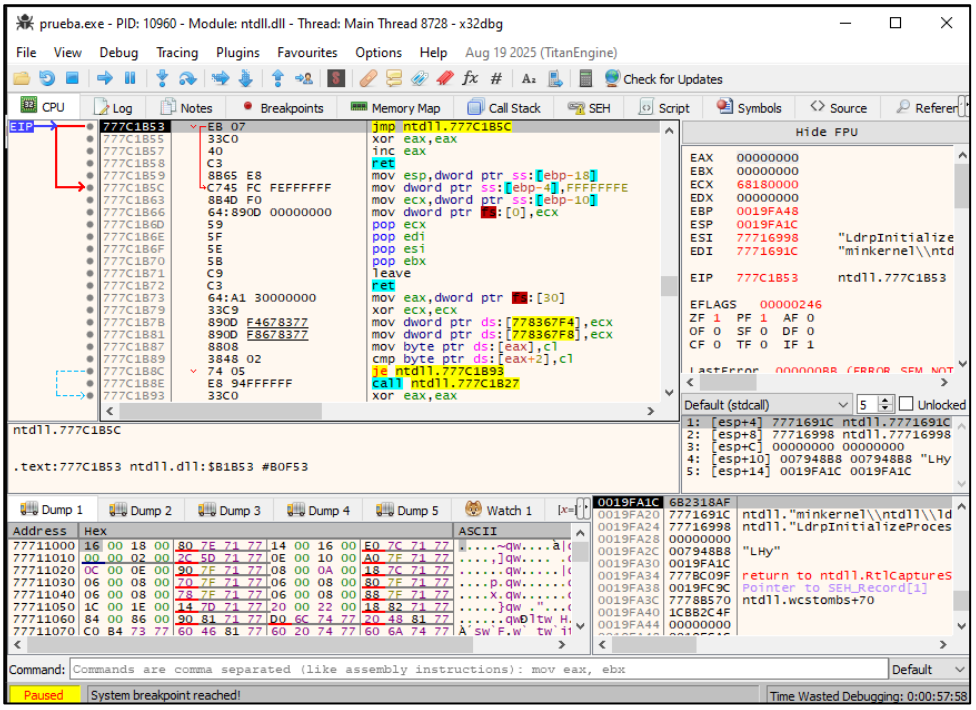
Introducción

El presente documento describe el proceso de análisis y modificación de un ejecutable antiguo, perteneciente a una aplicación crítica cuyo soporte ha desaparecido debido a la desaparición de la empresa desarrolladora. La finalidad de este trabajo es generar un parche ético que permita el funcionamiento continuo del software en un entorno productivo, específicamente burlando la validación de una clave de acceso (serial key) mediante técnicas de ingeniería inversa. Para ello, se emplea un depurador que facilita la inspección del código binario, localizando cadenas de texto como "Acceso" o "Error" que revelan los puntos de control. Una vez identificada la instrucción de comparación (CMP) y el salto condicional asociado (JNE), se procede a modificar el byte que define la condición, transformando el salto en uno que siempre conceda el acceso.

Posteriormente, se aplica un software de diferencias para verificar los cambios realizados y se desarrolla un programa en Python que automatiza la modificación, permitiendo replicar el parche de manera sencilla. El objetivo principal es documentar cada etapa del proceso, desde la depuración hasta la creación del script, con el fin de proporcionar una solución práctica y comprensible para preservar la operatividad de sistemas heredados.

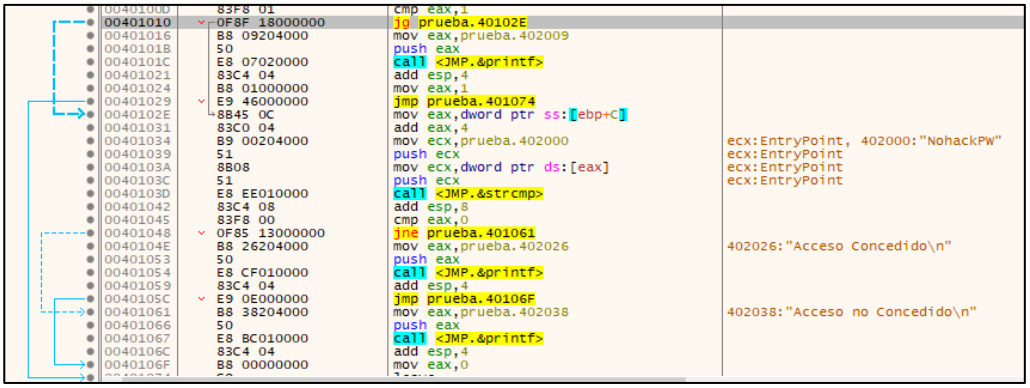
Paso 1

Abrir el binario en el depurador (Debugger).



Paso 2

Rastrear las cadenas de texto (Strings) para encontrar el mensaje "Error" o "Acceso".

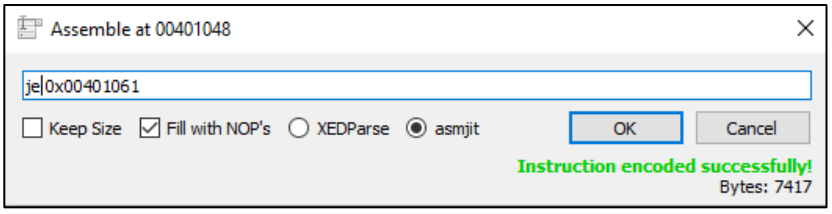


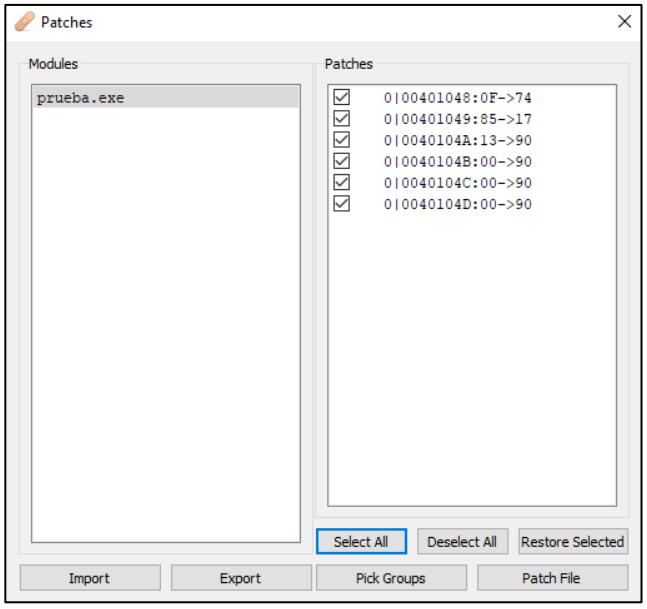
Paso 3

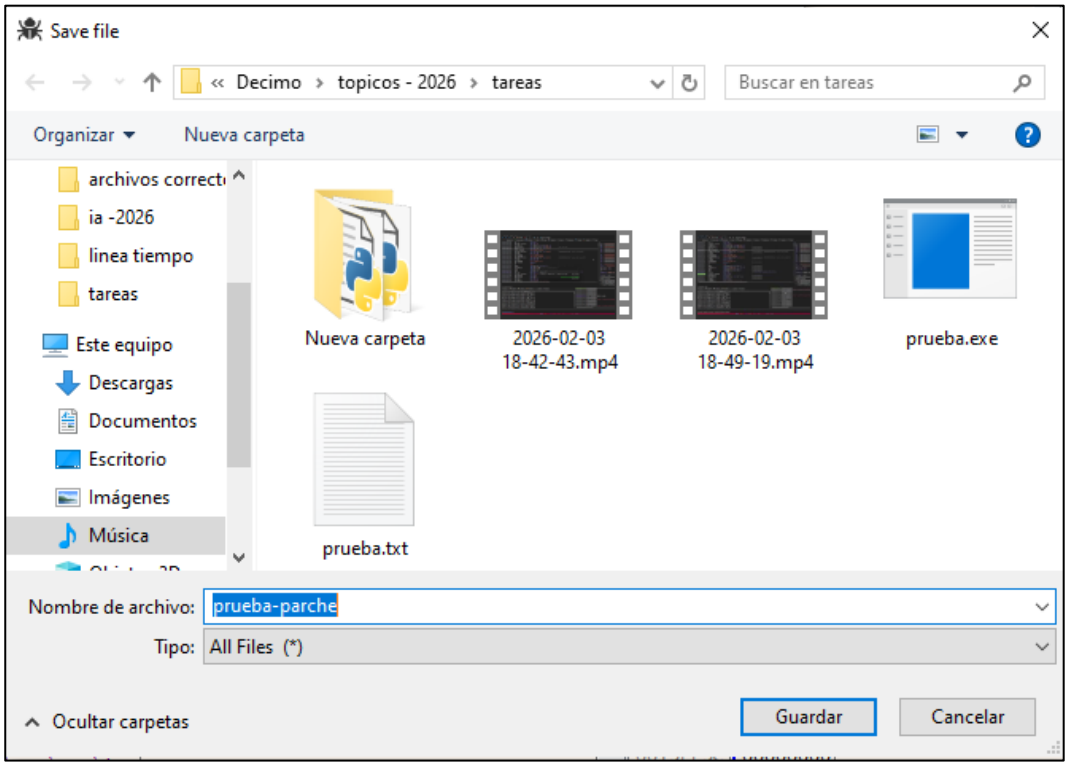
Identificar la instrucción de comparación (CMP) y el salto condicional (JE / JNE) que decide si la clave es correcta.

Paso 4

Guarda el ejecutable modificado.

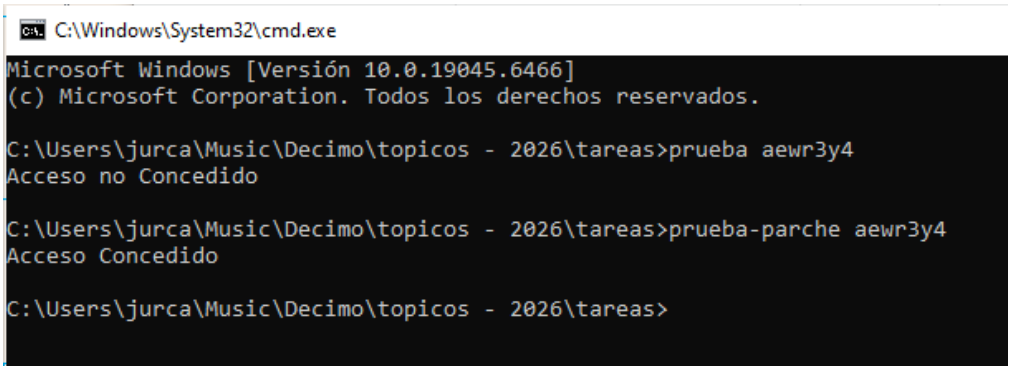






Paso 5

Aplica software de diferencias de archivos en el ejecutable original y en el modificado.



Paso 6

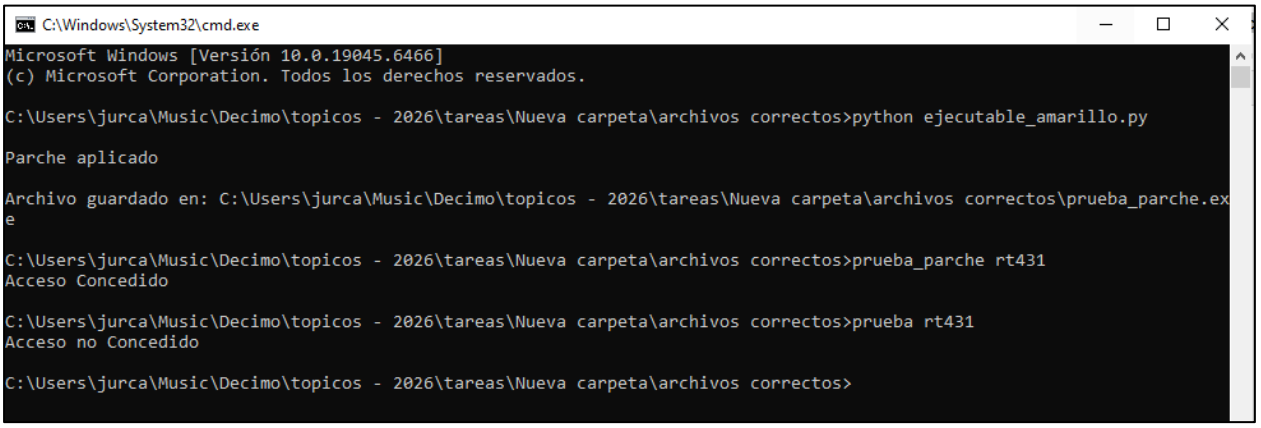
Identifica el cambio y realizar un programa en python que lea el ejecutable original y aplique los cambios pertinentes.

```python
from pathlib import Path

posicion = 0x248    # Lugar exacto donde comienza la instrucción
archivo_original = "prueba.exe"
archivo_parchado = "prueba_parche.exe"

carpeta = Path(__file__).parent
ruta_original = carpeta / archivo_original
ruta_parchado = carpeta / archivo_parchado

if not ruta_original.exists():
    print("\nNo se encuentra el archivo", archivo_original)
    exit()

with open(ruta_original, "rb") as archivo:
    datos = bytearray(archivo.read())

if posicion + 1 >= len(datos):
    print("\nLa posición está más allá del final del archivo.")
    exit()

primer_byte = datos[posicion]
segundo_byte = datos[posicion + 1]        # Debería ser 0x85 (JNE)

# ---- APLICAR EL PARCHE (JNE -> JE) -> cambiar el segundo byte: 0x85 ->
datos[posicion + 1] = 0x84

with open(ruta_parchado, "wb") as archivo:
    archivo.write(datos)

print("\nParche aplicado")
print(f"\nArchivo guardado en: {ruta_parchado}")
```

El programa realiza las siguientes acciones:

1. Verifica que el archivo prueba.exe existe en la misma carpeta.
2. Lee todo el contenido del archivo en modo binario.
3. Comprueba que la posición 0x248 (offset) esté dentro del archivo.
4. Modifica el byte en la posición 0x249 (segundo byte de la instrucción) cambiando su valor de 0x85 a 0x84, transformando un salto JNE en JE.
5. Guarda el archivo modificado como "prueba_parche.exe" y muestra un mensaje de confirmación.

A continuación se muestra prueba de su funcionamiento:



Conclusión

En conclusión, como equipo, hemos aprendido que la ingeniería inversa no solo es una herramienta de análisis de malware, sino también un recurso valioso para resolver problemas de software huérfano. A lo largo de este trabajo, nos enfrentamos al reto de entender la estructura binaria de un ejecutable, identificar la lógica de validación y modificarla sin alterar el resto del programa. Descubrimos que un simple cambio de un byte puede invertir por completo el flujo de decisión, convirtiendo un acceso denegado en un acceso concedido bajo cualquier contraseña. Esta experiencia nos hizo reflexionar sobre la responsabilidad ética que implica modificar software ajeno, aunque en este caso se trata de una necesidad legítima de producción, las mismas técnicas podrían ser mal utilizadas. Por ello, resaltamos la importancia de aplicar estos conocimientos con integridad y siempre con fines justificados. Además, comprendemos cómo la ciberseguridad se beneficia del entendimiento profundo del código, ya que permite detectar vulnerabilidades, crear parches de emergencia o analizar comportamientos maliciosos. En definitiva, este proyecto nos brindó una visión práctica de la relación entre la programación de bajo nivel, la depuración y la seguridad informática, habilidades que sin duda serán de gran utilidad en nuestra formación profesional.