



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.....	3
Paso 1	4
Paso 2	4
Paso 3.....	5
Paso 4.....	5
Paso 5.....	6
Paso 6.....	7
Conclusión	9

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).

The screenshot shows the Immunity Debugger interface with the following details:

- File View Debug Tracing Plugins Favourites Options Help Aug 19 2025 (TitanEngine)**
- CPU** pane: Assembly code for address **777C1B53**. The code includes `jmp ntdll.777C1B5C`, `ret`, and `je ntdll.777C1B93`.
- Registers** pane: Shows registers EAX, ECX, EDX, EBP, ESP, ESI, and EDI with their current values.
- Dump** pane: Shows memory dump information for address **777C1B5C**.
- Registers** pane: Shows the **EIP** register value **777C1B53**.
- Status Bar**: Paused, System breakpoint reached!, Time Wasted Debugging: 0:00:57:58

Paso 2

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

The screenshot shows the Immunity Debugger interface with the following details:

- Strings** pane: Displays several strings:
 - 00401000: 83F8 01
 - 00401010: v 0F8F 18000000
 - 00401016: B8 09204000
 - 00401018: 50
 - 0040101C: E8 07020000
 - 00401024: B8 01000000
 - 00401029: v E9 46000000
 - 0040102E: ^8845 OC
 - 00401031: B3C0 04
 - 00401034: B9 00204000
 - 00401039: 51
 - 0040103C: B808
 - 0040103D: 51
 - 00401042: E8 E0100000
 - 00401045: B3C4 08
 - 00401048: v OF85 13000000
 - 0040104E: B8 26204000
 - 00401053: 50
 - 00401054: E8 C0100000
 - 00401059: v E9 0E000000
 - 00401061: B8 38204000
 - 00401066: 50
 - 00401067: E8 BC010000
 - 0040106C: B3C4 04
 - 0040106F: B8 00000000
- Registers** pane: Shows the **ecx** register with the value **402000: "NohackPW"**.
- Status Bar**: 402026: "Acceso Concedido\n", 402038: "Acceso no Concedido\n"



Paso 3

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

```

00401000  83F8 01      cmp eax,1
00401010  B8 09204000 mov eax,prueba.402009
00401016  50              push eax
00401018  E8 07020000 call <JMP.&printf>
0040101C  83C4 04      add esp,4
00401021  B8 01000000 mov eax,1
00401024  E9 46000000 jmp prueba.401074
00401029  8B45 0C      mov eax,dword ptr ss:[ebp+C]
0040102E  53 04          mov ecx,prueba.402000
00401034  B9 00204000 push ecx
00401039  51              mov eax,dword ptr ds:[eax]
0040103A  B808            push ecx
0040103C  51              push ecx
0040103D  E8 EE010000 call <JMP.&strcmp>
0040103F  83C4 08      add esp,8
00401042  B8 00000000 mov eax,0
00401045  53 00          mov ecx,prueba.401061
00401048  0F85 13000000 jne prueba.401061
0040104E  B8 26204000 mov eax,prueba.402026
00401053  50              push eax
00401054  E8 CF010000 call <JMP.&printf>
00401059  83C4 04      add esp,4
0040105C  B8 0E000000 jmp prueba.40106F
00401061  B8 38204000 mov eax,prueba.402038
00401066  50              push eax
00401067  E8 BC010000 call <JMP.&printf>
0040106C  83C4 04      add esp,4
0040106F  B8 00000000 mov eax,0

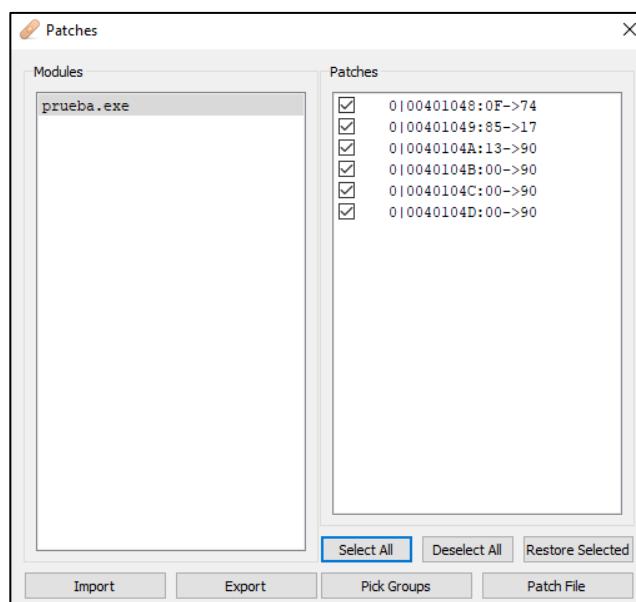
```

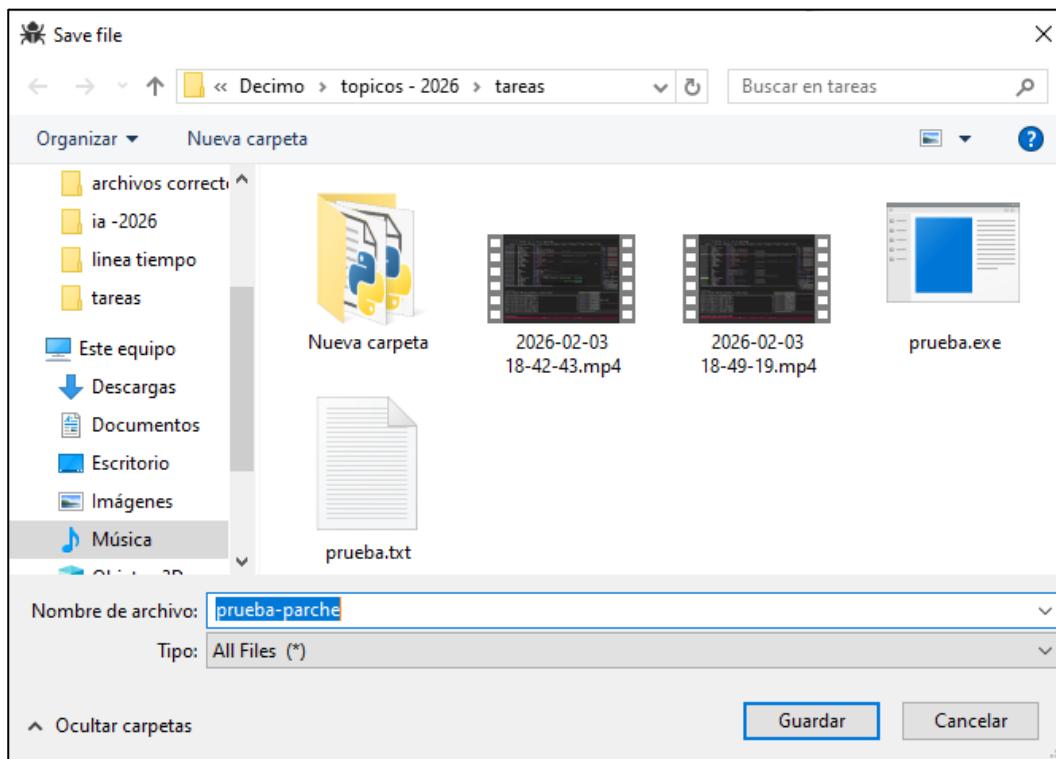
ecx:EntryPoint, 402000:"NohackPW"
 ecx:EntryPoint
 ecx:EntryPoint
 ecx:EntryPoint
 ecx:EntryPoint

402026:"Acceso Concedido\n"
 402038:"Acceso no Concedido\n"

Paso 4

Guarda el ejecutable modificado.





Paso 5

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

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19045.6466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas>prueba aewr3y4
Acceso no Concedido

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas>prueba-parche aewr3y4
Acceso Concedido

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas>
```

Paso 6

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

```
from pathlib import Path

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

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

# ---- APlicar El PARCHE (JNE -> JE) -> cambiar el segundo byte: 0x85 ->
datos[posición + 1] = 0x84

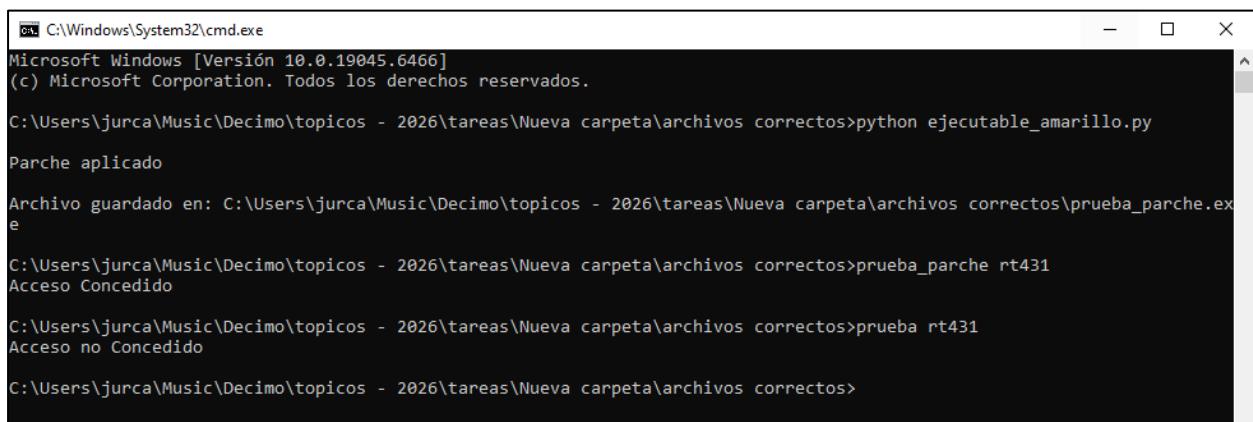
with open(ruta_parchado, "wb") as archivo:
    archivo.write(datos)

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:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19045.6466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas\Nueva carpeta\archivos correctos>python ejecutable_amarillo.py
Parche aplicado

Archivo guardado en: C:\Users\jurca\Music\Decimo\topicos - 2026\tareas\Nueva carpeta\archivos correctos\prueba_parche.exe

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas\Nueva carpeta\archivos correctos>prueba_parche rt431
Acceso Concedido

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas\Nueva carpeta\archivos correctos>prueba rt431
Acceso no Concedido

C:\Users\jurca\Music\Decimo\topicos - 2026\tareas\Nueva carpeta\archivos correctos>
```

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.