Pwning 2
Buffer Overflows

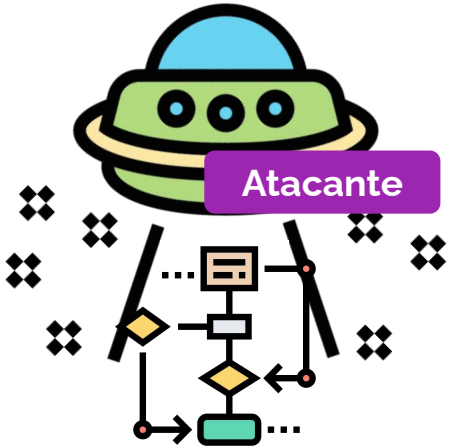
CC5325 - Taller de Hacking Competitivo

Ataques de Secuestro del Control de Flujo

Un programa suele ejecutar una secuencia de instrucciones en un orden determinado

Un atacante puede querer cambiar el orden de las instrucciones ejecutadas (o agregar nuevas)

A continuación veremos algunos tipos de bugs que permiten controlar el flujo de un programa a través de las entradas que recibe.



¿Cómo se ejecuta el código que escribo? *(lenguaje compilado)*

```
#include <stdio.h>
int main() {
     printf("cc5312");
      return 0;
}
```

```
...
leaq    .LC0(%rip), %rdi
xorl    %eax, %eax
call    printf@PLT
...
```

```
...
\x48\x31\xc9\x48\xf7\xe1\
x04\x3b\x48\xbb\x2f\x62\x
69\x6e\x2f\x2f\x73\x68\x5
2\x53\x54\x5f\x52\x57\x54
...
```

https://godbolt.org/

Completamente equivalentes

Código Fuente → **Código Assembly** ↔ **Código de máquina**

Instrucciones y Registros en ASM

Registros

Memoria de acceso rápido ubicadas en el procesador

De uso General

`(E)AX` `(E)BX` `(E)CX` `(E)DX`

Punteros

`EBP` `ESP` `EDI` `ESI` `EIP`

Instrucciones

Algunas instrucciones que conviene recordar

Operaciones

`add` `sub` `mul` `div` `and` `xor`

Pila

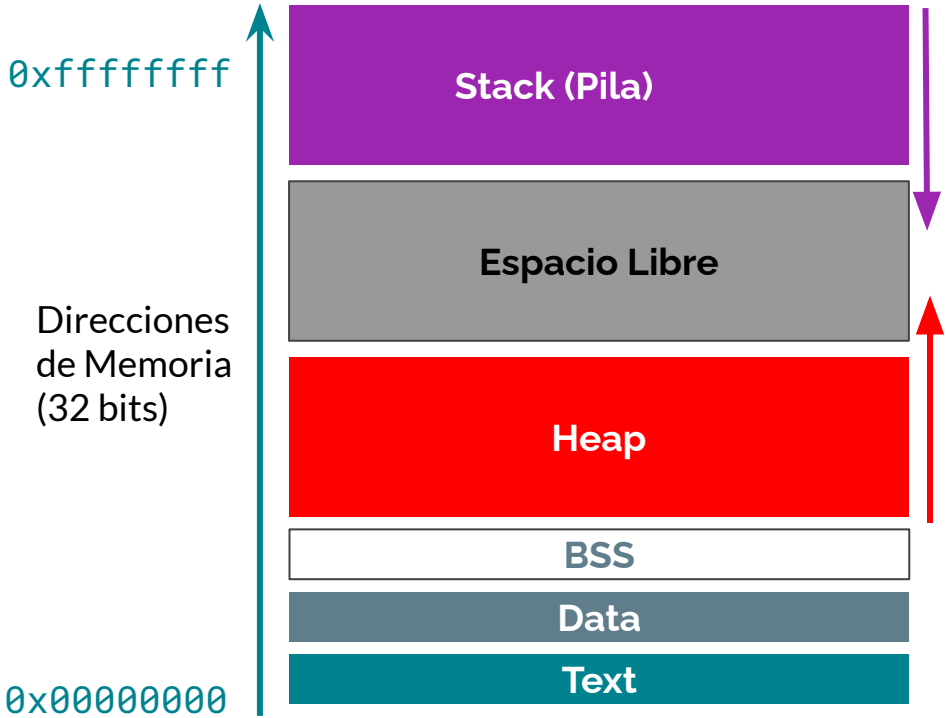
`push` `pop`

Control de flujo

`cmp` `je` `jne` `jg` `jl`

https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

Manejo de Memoria en grandes rasgos (en arquitectura x86)



Almacena las variables locales de cada llamado a una función

Usado por *Stack* y *Heap* según necesitan

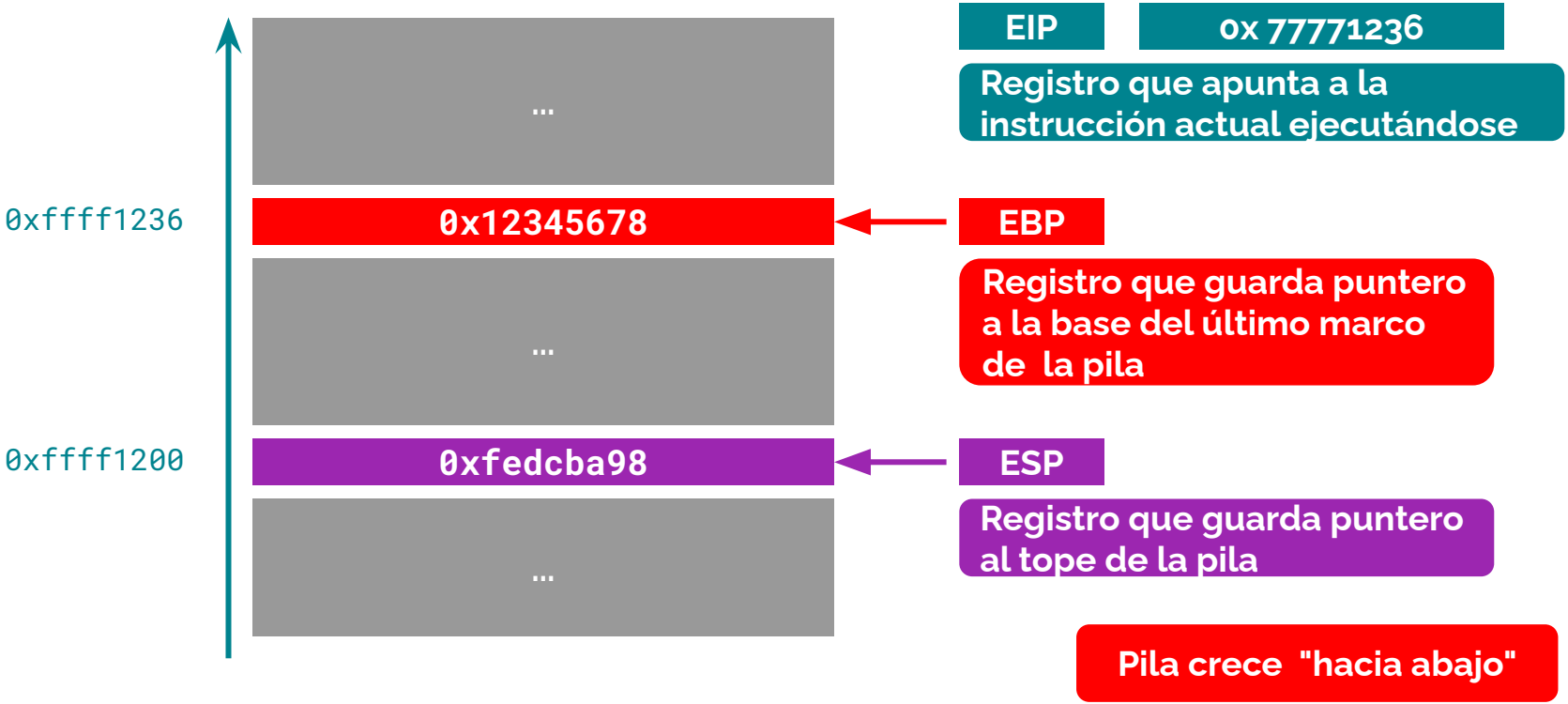
Memoria dinámica: Almacena variables declaradas con *malloc* (y otras funciones)

Variables globales no inicializadas

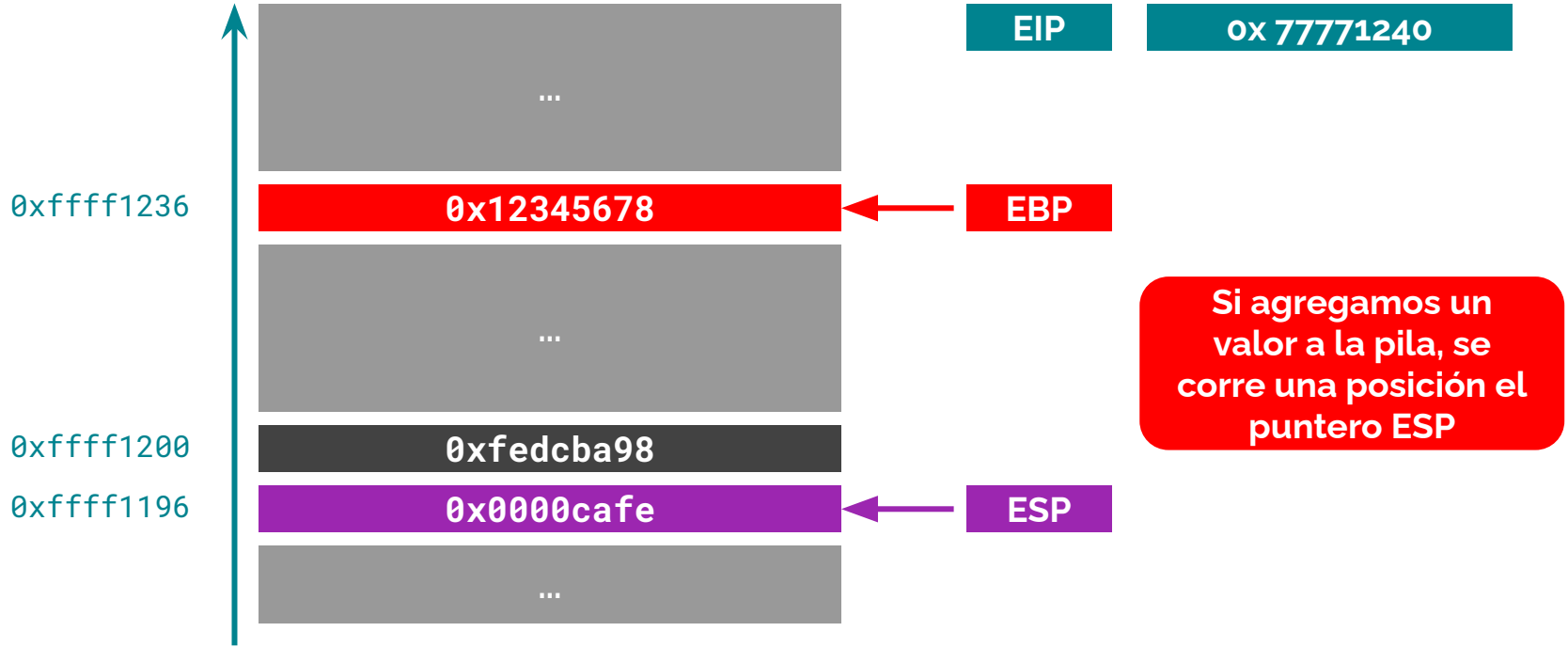
Constantes

Código a ejecutar

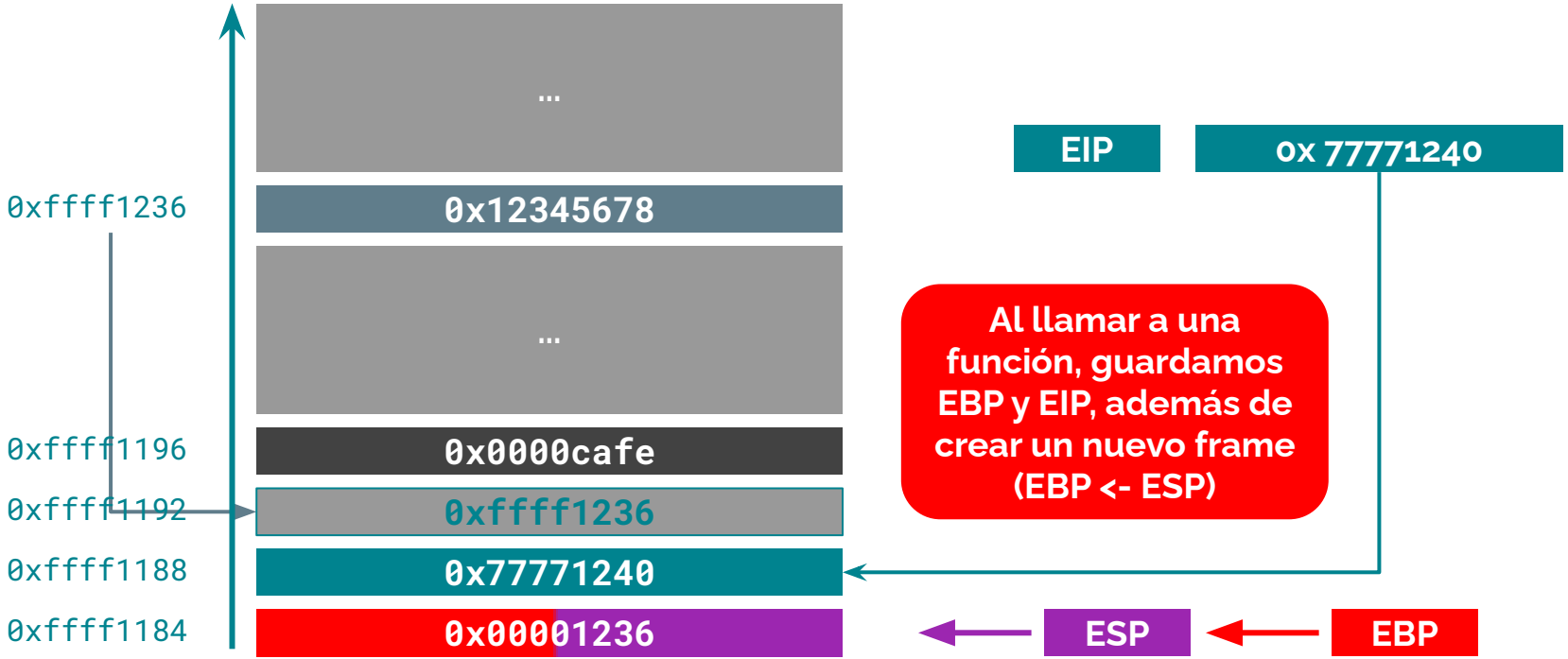
Estructura de la pila



Estructura de la pila



Estructura de la pila



Shellcode

Instrucciones son representables como datos

Código Assembly ⟷ **Código de máquina**

```
jmp    0x1f             # 2 bytes
popl   %esi             # 1 byte
movl   %esi,0x8(%esi)   # 3 bytes
xorl   %eax,%eax        # 2 bytes
movb   %eax,0x7(%esi)   # 3 bytes
movl   %eax,0xc(%esi)   # 3 bytes
movb   $0xb,%al         # 2 bytes
movl   %esi,%ebx        # 2 bytes
leal   0x8(%esi),%ecx   # 3 bytes
leal   0xc(%esi),%edx   # 3 bytes
int    $0x80            # 2 bytes
xorl   %ebx,%ebx        # 2 bytes
movl   %ebx,%eax        # 2 bytes
inc    %eax             # 1 bytes
int    $0x80            # 2 bytes
call   -0x24            # 5 bytes
.string "/bin/sh"       # 8 bytes
```

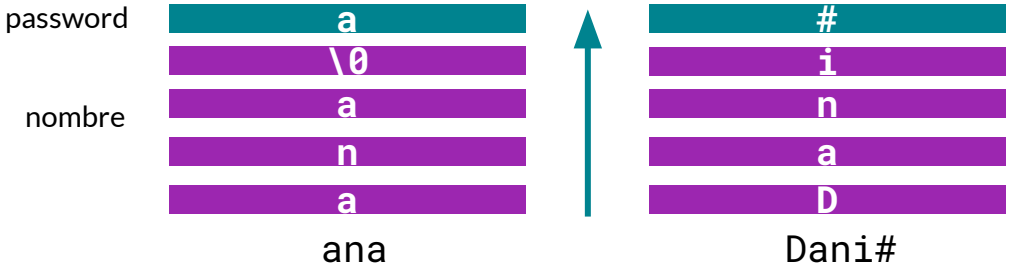
```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x4
6\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x8
0\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bi
n/sh";
```

http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

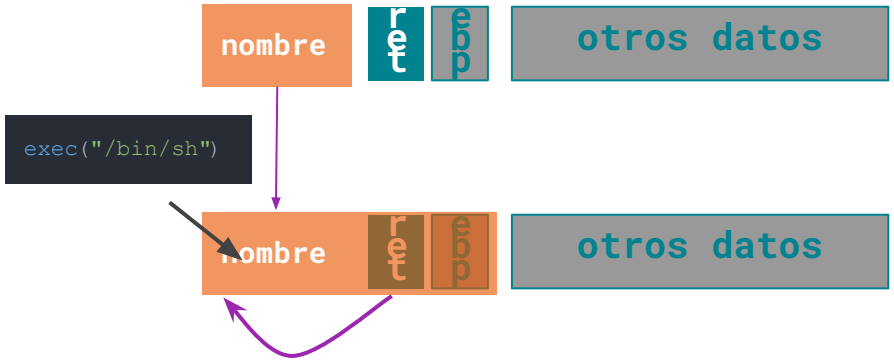
Buffer Overflow

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("usage: %s name", argv[0]);
        exit(1);
    }
    char password = 'a';
    char  nombre[4];
    strcpy(nombre, argv[1]);
    printf("hola %s. Veremos si tienes acceso...\n",
nombre);
    if (password == '#') {
        printf("Acceso otorgado! (password=%c)\n",
password);
    } else {
        printf("Acceso denegado (password=%c)\n",
password);
    }
    exit(0);
}
```

Sobreescribir datos

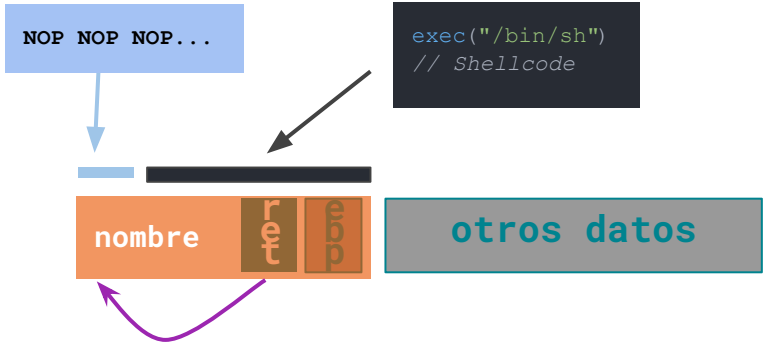


¿Y si sobreescribimos más allá?



¿Qué hago si no conozco **exactamente** la dirección de *ret*?

NOP Slide



Rellenar con instrucciones NOP (No Operation) previas al programa que queremos ejecutar, de forma que si el puntero *ret* apunta a una de ellas, igual se llegue sin problemas al inicio de nuestro código.

Prevención de ejecución de código

Write xor Execute (W^X)

Marcar segmentos de pila como no ejecutables

Limitación: Algunas aplicaciones requieren un Stack ejecutable

PaX (Linux)

DEP (Windows)

Aleatorización de ubicación de elementos en memoria

Dificulta "adivinar" dónde se ubica cierto código ejecutable de una librería o valor secreto

Instruction Set Randomization

Syscall Randomization

Address Space Layout Randomization

Detección en tiempo de ejecución

Canarios (StackGuard, PointGuard)

Si intento modificar *ret*, pasaré a llevar el canario

Programa revisa que canario tenga valor esperado antes de salir de frame

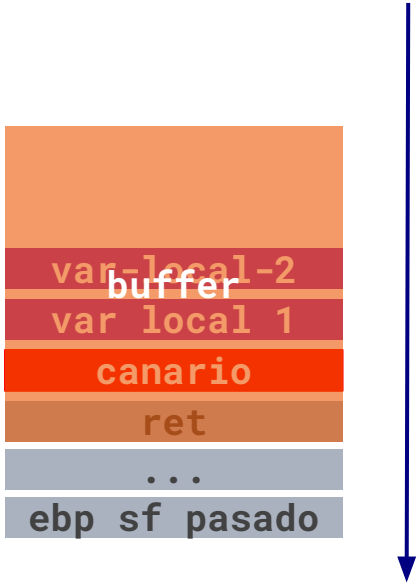
flag -fstack-protector

Canario Al azar

Elegido al partir programa

Canario Terminator

Contiene "\x00", lo que impide seguir leyendo memoria como si fuera str.



https://sourceforge.net/projects/safeclib/

¿Cómo ejecutar un BO en CTFs?

- Esto lo veremos en la clase.
- Si no pueden ir a la clase, revisen https://padraignix.github.io/reverse-engineering/2019/09/28/buffer-overflow-practical-case-study/

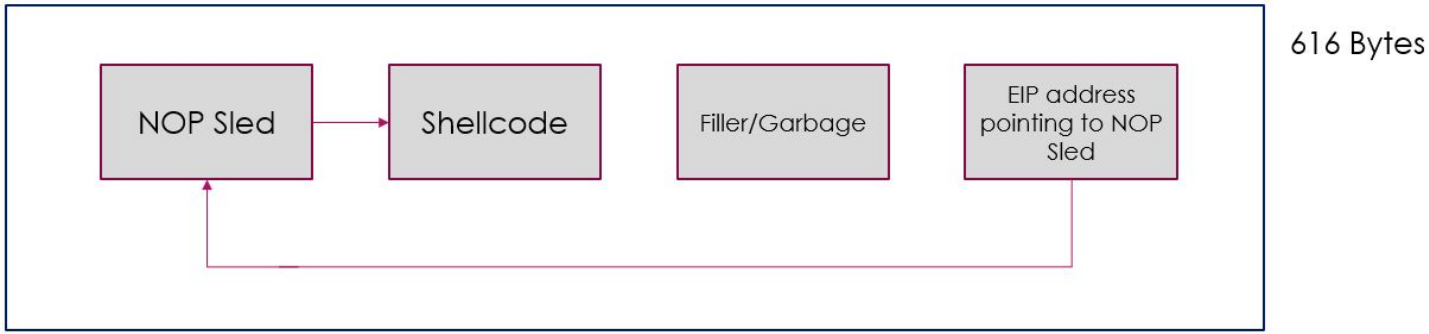


Imagen del post de padraignix