

#CátedrasCiber

Módulo IV: Ingeniería inversa y Explotación de binarios

13/11/2024



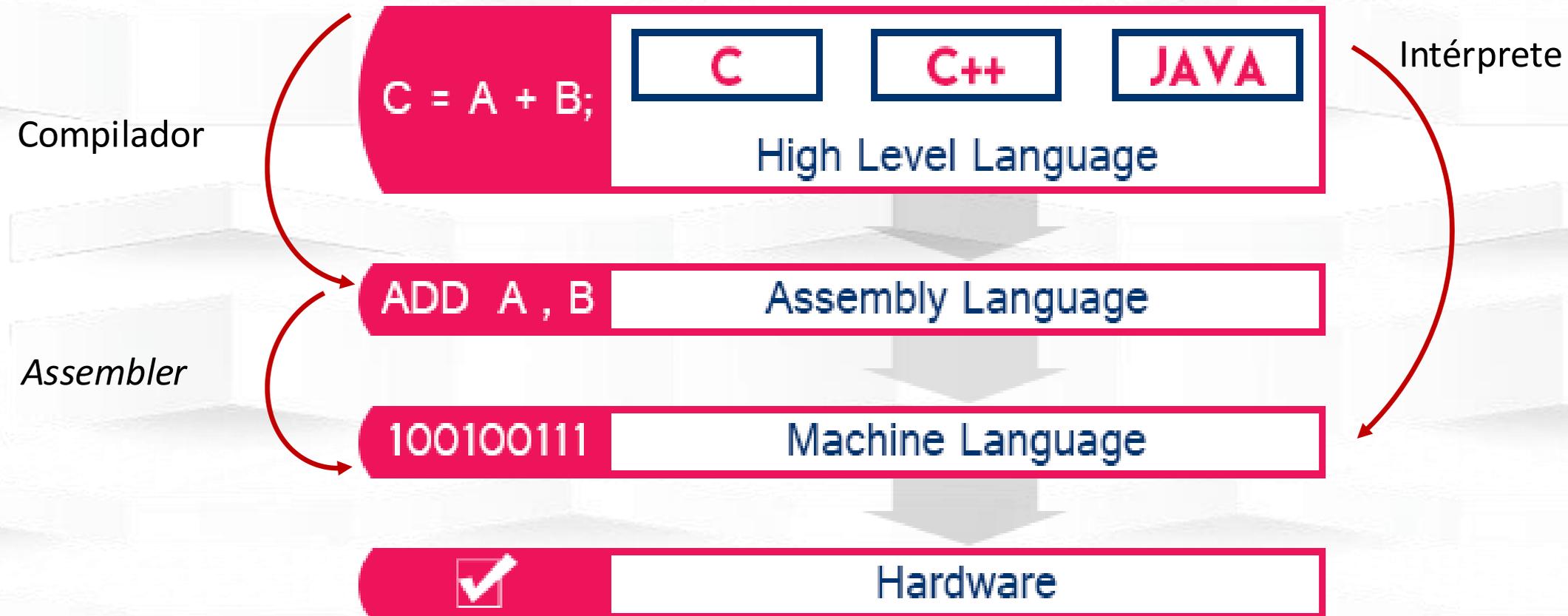
Ingeniería Inversa

¿Qué es la ingeniería inversa?

- La ingeniería inversa, *reverse engineering* o reversing implica seguir el procedimiento de fabricación de un producto **en el sentido inverso al habitual**.
- Trataremos de obtener información sobre el funcionamiento interno de un archivo sin tener datos reales sobre su estructura.
 - Por ejemplo, a partir de un .exe o ELF.



Proceso de compilado/interpretado



Requisitos para hacer reversing

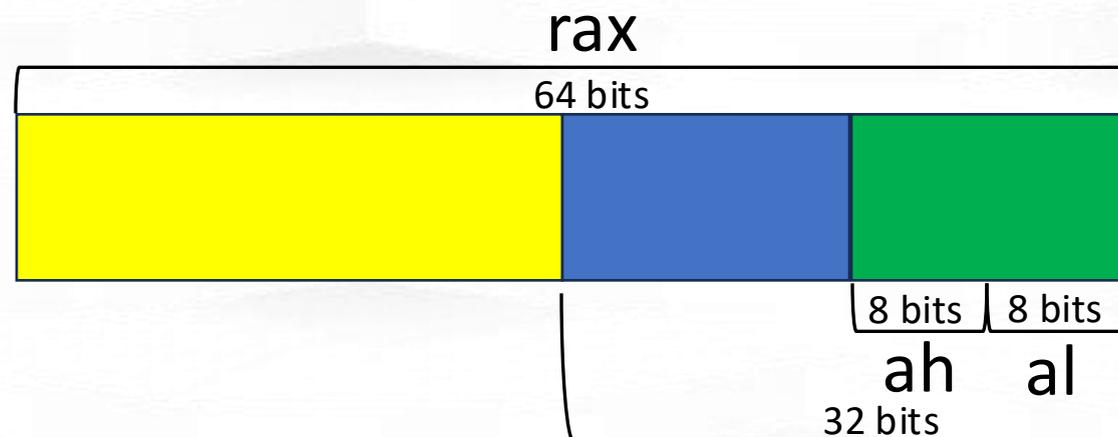
Para hacer reversing a un código, necesitaremos varias herramientas:

- Conocimientos a nivel de la arquitectura (bajo nivel) para la que se compiló el código (diseño del microprocesador, repertorio de instrucciones, etc.)
- Conocimientos de herramientas que nos ayuden en este proceso: desensambladores, decompiladores, analizadores de código (estáticos y dinámicos), depuradores (debuggers), etc.

Requisitos para hacer reversing

Debéis conocer los siguientes registros del procesador:

- RAX, RBX, RCX, RDX: Son registros de propósito general del procesador, el acumulador, la base, el contador y el registro de datos, respectivamente. Pueden guardar tanto datos como direcciones.
- RBP, RSP: Son los registros de puntero de base y de pila.
- RIP: Es el contador del programa, que almacena la dirección de la siguiente instrucción a ejecutar.



Ingeniería inversa - *Debugging*

Asumiendo que no tenemos acceso directo al código fuente, tenemos dos formas principales de comenzar las tareas de reversing:

- Análisis estático: intentamos recuperar la máxima información posible del código fuente de la aplicación para conocer cómo funciona.
- Análisis dinámico: ejecutamos la aplicación (**SIEMPRE EN UNA VM**) y observamos su comportamiento ante diferentes entradas.

Como complemento al análisis dinámico, podemos utilizar *debuggers* o depuradores.

Ingeniería inversa - *Debugging*

- Quizás el *debugger* más utilizado en el ámbito de la ciberseguridad sea GDB.
- Permite trabajar desde la línea de comandos o emplear alguna de sus interfaces.
 - Como GDB TUI (interfaz en modo texto) o Insight (interfaz gráfico).
 - También se puede utilizar alguno de sus front-ends, como gdbgui, Nemiver, WinGDB o el propio Emacs.
- Hoy en día existen complementos muy potentes y útiles para la línea de comandos.
 - Ejemplos de ello son [gef](#), [peda](#) o [pwndbg](#).
 - Puedes instalar todos con [esta herramienta](#).

Ingeniería inversa - *Debugging*

Con estas herramientas podemos observar el código ensamblador que se ejecuta cuando se lanza una aplicación, qué funciones se llaman, el contenido de la memoria y de los registros del procesador.

También se pueden introducir *breakpoints* para ir parando la ejecución cuando sea necesario.

Todo esto sin necesidad de tener disponible el código fuente de la aplicación analizada, basta con disponer del ejecutable.

Ingeniería inversa - *Debugging*

Existen varias herramientas para realizar las tareas de decompilado/desensamblado:

- Ghidra
- Binary Ninja
- IDA (Free o pro)
- Olly Dbg
- x32debugger/x64debugger
- Radare2 (Cutter)



Alto Nivel → Bajo Nivel → Máquina
Compilación -- Ensamblado

Ejemplos

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int variable=0;
7
8     while(variable!=13)
9     {
10        printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11        scanf("%d",&variable);
12    }
13    printf("Correcto! URJC{Correctp}");
14
15    return 0;
16 }
17
```

```
.....
*                               FUNCTION
*                               *
.....
int __fastcall __main(void)
EAX:4 <RETURN>
__main XREF[3]: __tmainCRTStartup:004013b6(c),
main:00401538(c), 00405108(*)
= ??

00402110 8b 05 2a MOV EAX,dword ptr [initialized]
52 00 00

00402116 85 c0 TEST EAX,EAX
00402118 74 06 JZ LAB_00402120
0040211a f3 c3 RET
0040211c 0f ?? 0Fh
0040211d 1f ?? 1Fh
0040211e 40 ?? 40h @
0040211f 00 ?? 00h

LAB_00402120 XREF[1]: 00402118(j)
= ??
00402120 c7 05 16 MOV dword ptr [initialized],0x1
52 00 00
01 00 00 00
0040212a eb 84 JMP __do_global_ctors

DAT_0040212c XREF[1]: 0040510c(*)
0040212c 90 ?? 90h
0040212d 90 ?? 90h
0040212e 90 ?? 90h
0040212f 90 ?? 90h

.....
*                               FUNCTION
*                               *
.....
void __fastcall __security_init_cookie(void)
```

Ejemplos

```
1
2 int main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     int local_c;
6
7     _main();
8     local_c = 0;
9     while (local_c != 0xd) {
10         printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11         scanf("%d",&local_c);
12     }
13     printf("Correcto! URJC{Correctp}");
14     return 0;
15 }
16
```

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int variable=0;
7
8     while(variable!=13)
9     {
10         printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11         scanf("%d",&variable);
12     }
13     printf("Correcto! URJC{Correctp}");
14
15     return 0;
16 }
17
```

No siempre se trata de decompilar

- A veces, habrá retos dentro de la categoría de reversing que consistirán en análisis de código fuente o, dado un código ofuscado, tratar de recuperar el código original.
 - O, al menos, una versión más legible

```
var _0x2da72b=_0x23ae;function _0x23ae(_0x384bcb,_0x4d943a){var
_0x56dc43=_0x56dc();return
_0x23ae=function(_0x23ae43,_0x4d79f2){_0x23ae43=_0x23ae43-
0x143;var _0x157b9f=_0x56dc43[_0x23ae43];return
_0x157b9f;},_0x23ae(_0x384bcb,_0x4d943a);}function _0x56dc(){var
_0x2de2d7=['8101179LDLJdd','charAt','81306hLMcQW','65oKTNMq','
10CiIRRV','1431062rTFndF','5934680LMAxTh','URJC{Desofuscando}','7
09227QTsrkv','3969945atiJhi','1030616LcvUIK','9YcTpzQ'];_0x56dc=fun
ction(){return _0x2de2d7;};return
_0x56dc();}(function(_0x2820fc,_0x53eb87){var
_0x2064a7=_0x23ae,_0x37c6cf=_0x2820fc();while(!![]){try{var
_0x4c24c9=-parseInt(_0x2064a7(0x144))/0x1+-
parseInt(_0x2064a7(0x14d))/0x2+-parseInt(_0x2064a7(0x147))/0x3*(-
parseInt(_0x2064a7(0x146))/0x4)+parseInt(_0x2064a7(0x14b))/0x5*(
parseInt(_0x2064a7(0x14a))/0x6)+-
parseInt(_0x2064a7(0x145))/0x7+parseInt(_0x2064a7(0x14e))/0x8+pa
rseInt(_0x2064a7(0x148))/0x9*(parseInt(_0x2064a7(0x14c))/0xa);if(_0
x4c24c9===_0x53eb87)break;else
_0x37c6cf['push'](_0x37c6cf['shift']());}catch(_0x22350c){_0x37c6cf['p
ush'](_0x37c6cf['shift']());}}(_0x56dc,0x9249e));var
numero=0x1,flag=_0x2da72b(0x143);while(numero>=0x0){numero=pr
ompt('introduce\x20un\x20número',''),secreta(flag,numero);}function
secreta(_0x18b49d,_0x34c2bb){var
_0x3bc3be=_0x2da72b;alert(_0x18b49d[_0x3bc3be(0x149)](_0x34c2
bb));}
```

<https://deobfuscate.io/>

No siempre se trata de decompilar

- A veces, habrá retos dentro de la categoría de reversing que consistirán en análisis de código fuente o, dado un código ofuscado, tratar de recuperar el código original.
 - O, al menos, una versión más legible

```
var _0x2da72b=_0x23ae;function _0x23ae(_0x384bcb,_0x4d943a){var
_0x56dc43=_0x56dc();return
_0x23ae=function(_0x23ae43,_0x4d79f2){_0x23ae43=_0x23ae43-
0x143;var _0x157b9f=_0x56dc43[_0x23ae43];return
_0x157b9f;},_0x23ae(_0x384bcb,_0x4d943a);}function _0x56dc(){var
_0x2de2d7=["8101179LDLJdd','charAt','81306hLMcQW','65oKTNMq','
10CiIRRV','1431062rTFndF','5934680LMAxTh','URJC{Desofuscando}','7
09227QTsrkv','3969945atiJhi','1030616LcvUIK','9YcTpzQ'];_0x56dc=fun
ction(){return _0x2de2d7;};return
_0x56dc();}(function(_0x2820fc,_0x53eb87){var
_0x2064a7=_0x23ae,_0x37c6cf=_0x2820fc();while(![]){try{var
_0x4c24c9=-parseInt(_0x2064a7(0x144))/0x1+-
parseInt(_0x2064a7(0x14d))/0x2+-parseInt(_0x2064a7(0x147))/0x3*(-
parseInt(_0x2064a7(0x146))/0x4)+parseInt(_0x2064a7(0x14b))/0x5*(
parseInt(_0x2064a7(0x14a))/0x6)+-
parseInt(_0x2064a7(0x145))/0x7+parseInt(_0x2064a7(0x14e))/0x8+pa
rseInt(_0x2064a7(0x148))/0x9*(parseInt(_0x2064a7(0x14c))/0xa);if(_0
x4c24c9===_0x53eb87)break;else
_0x37c6cf['push'](_0x37c6cf['shift']());}catch(_0x22350c){_0x37c6cf['p
ush'](_0x37c6cf['shift']());}}(_0x56dc,0x9249e));var
numero=0x1,flag=_0x2da72b(0x143);while(numero>=0x0){numero=pr
ompt('introduce\x20un\x20número',''),secreta(flag,numero);}function
secreta(_0x18b49d,_0x34c2bb){var
_0x3bc3be=_0x2da72b;alert(_0x18b49d[_0x3bc3be(0x149)](_0x34c2
bb));}
```

<https://deobfuscate.io/>

```
var numero = 0x1;
while (numero >= 0x0) {
    numero = prompt("introduce un número", "");
    secreta("URJC{Desofuscando}", numero);
}
function secreta(_0x18b49d, _0x34c2bb) {
    alert(_0x18b49d.charAt(_0x34c2bb));
}
```

Análisis dinámico: ltrace y strace

- Si es necesario pasar al análisis dinámico, hay ciertas herramientas que pueden ayudarnos a comprender cuál es el funcionamiento del binario.
- Dos de las más útiles son **ltrace** y **strace**

STRACE

Ejecuta el programa hasta que termina

Intercepta las llamadas **al sistema**

También intercepta las señales que recibe el programa

LTRACE

Ejecuta el programa hasta que termina

Intercepta las llamadas **dinámicas a librerías**

También intercepta las señales que recibe el programa

Ejemplo strace y ltrace

- Programa sencillo en C (hello.c).
- Veamos las diferencias de comportamiento de strace y ltrace.

```
File: hello.c
1  #include <stdio.h>
2
3  int main(){
4      printf("%s\n","Hello world!");
5  }
```

Ejemplo strace y ltrace

```
execve("./hello", [ "./hello" ], 0x7fff33825ad0 /* 40 vars */) = 0
brk(NULL) = 0x5571c4413000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=44313, ...}) = 0
mmap(NULL, 44313, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7eff26fe3000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\177\2\0\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839168, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7eff26fe1000
mmap(NULL, 1852480, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7eff26e1c000
mprotect(0x7eff26e42000, 1658880, PROT_NONE) = 0
mmap(0x7eff26e42000, 1347584, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7eff26e42000
mmap(0x7eff26f8b000, 307200, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16f000) = 0x7eff26f8b000
mmap(0x7eff26fd7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7eff26fd7000
mmap(0x7eff26fdd000, 13376, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7eff26fdd000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7eff26e1a000
arch_prctl(ARCH_SET_FS, 0x7eff26fe2540) = 0
mprotect(0x7eff26fd7000, 12288, PROT_READ) = 0
mprotect(0x5571c2d08000, 4096, PROT_READ) = 0
mprotect(0x7eff27018000, 4096, PROT_READ) = 0
munmap(0x7eff26fe3000, 44313) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
brk(NULL) = 0x5571c4413000
brk(0x5571c4434000) = 0x5571c4434000
write(1, "Hello world!\n", 13Hello world!
) = 13
exit_group(0) = ?
+++ exited with 0 +++
```

STRACE

LTRACE

```
puts("Hello world!"Hello world!
)
+++ exited (status 0) +++
```

= 13

Ejemplo strace y ltrace

```
1  #include <stdio.h>
2
3  int main() {
4      char frase[50];
5
6      printf("%s", "Introduce la password: ");
7      scanf("%s", &frase);
8
9      if (strcmp(frase, "impossible_password") == 0){
10         printf("%s\n", "Enhorabuena campeón");
11     } else {
12         printf("%s\n", "Oh! que pena... no era esa");
13     }
14 }
```

```
> ltrace ./pass
printf("%s", "Introduce la password: ")           = 23
__isoc99_scanf(0x558199c6701c, 0x7fffb7c64e10, 0, 0Introduce la password: medaigual
)
                                                    = 1
strcmp("medaigual", "impossible_password")        = 4
puts("Oh! que pena... no era esa"0h! que pena... no era esa
)
                                                    = 27
+++ exited (status 0) +++
```

Ejemplo strace y ltrace

- El argumento "-e" sirve para especificar el nombre de una llamada a librería/sistema. De esta manera solo se recogería el output de dicha llamada.

```
> ltrace -e strcmp ./pass
Introduce la password: prueba
pass->strcmp("prueba", "impossible_password") = 7
Oh! que pena... no era esa
+++ exited (status 0) +++
```

- El argumento -i imprime a su vez la dirección de la instrucción que se está ejecutando

```
> ltrace -i -e strcmp ./pass
Introduce la password: prueba
[0x55bc9cd071b0] pass->strcmp("prueba", "impossible_password") = 7
Oh! que pena... no era esa
[0xffffffffffffffff] +++ exited (status 0) +++
```

Ejemplo strace y ltrace

- El argumento "--output=<fichero>" sirve para especificar la ruta del output en caso de que queramos que se guarde.

```
> ltrace -i -e strcmp --output=mi_output ./pass
Introduce la password: prueba
Oh! que pena... no era esa
> cat mi_output
```

	File: mi_output
1	[0x560369fbf1b0] pass->strcmp("prueba", "impossible_password") = 7
2	[0xffffffffffffffff] +++ exited (status 0) +++

Explotación de binarios

Introducción al *exploiting* (pwn)

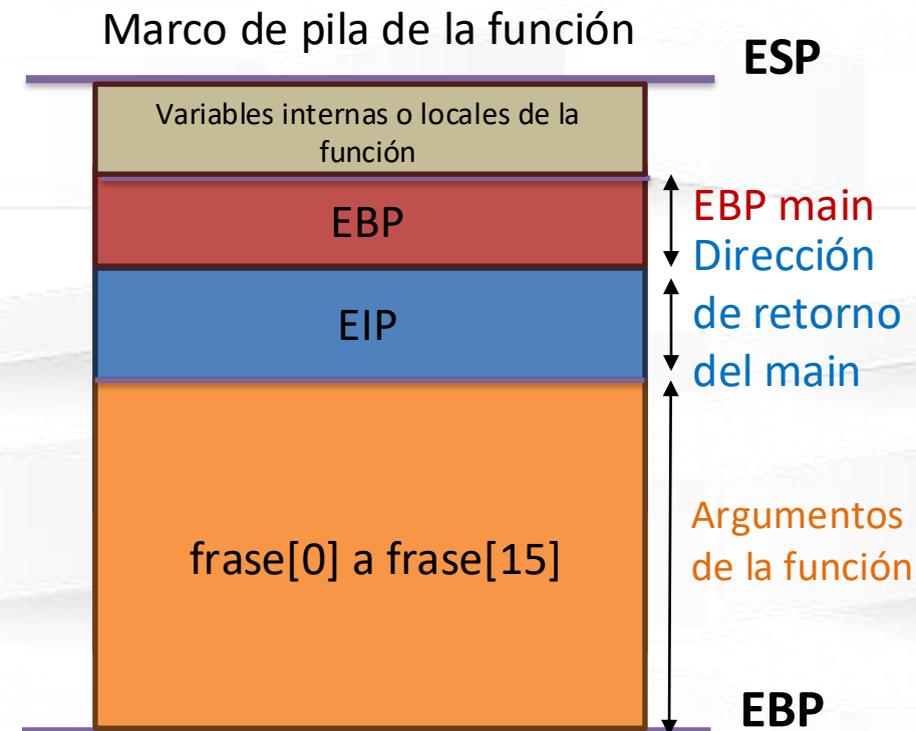
- La explotación de binarios está estrechamente relacionada con la ingeniería inversa.
 - Para tener éxito en la explotación, hay que analizar y entender el código fuente de la aplicación, de manera que encontremos puntos vulnerables en el mismo.
- Es requisito indispensable entender cómo funciona la pila del sistema, los registros del procesador y tener conocimientos de la arquitectura.

Introducción al *exploiting* (pwn)

- Es un campo demasiado amplio como para entrar en detalle.
- Daremos nociones básicas para realizar explotaciones simples.
- La explotación más básica, lo cual no implica que sea sencilla de explotar en determinados contextos, es *buffer overflow*.

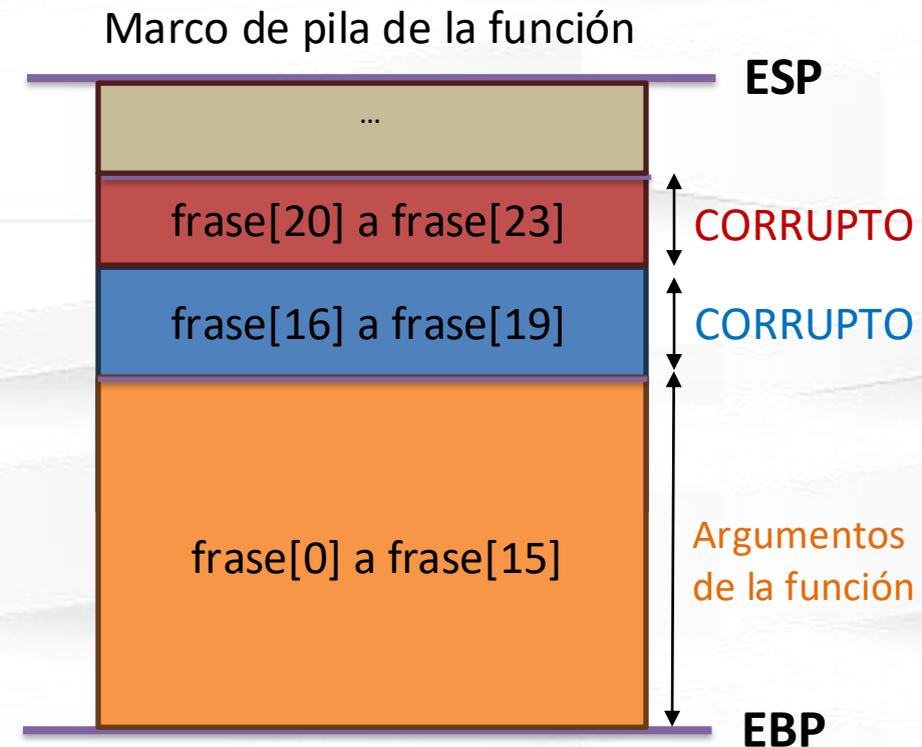
Buffer Overflow

```
void copia_cadena(char *cadena) {  
    char buffer[16];  
    strcpy(buffer, cadena);  
}  
  
void main() {  
    char frase[16];  
    int i;  
    for(i = 0; i < 15; i++) {  
        frase[i] = 'M';  
    }  
    copia_cadena(frase);  
}
```



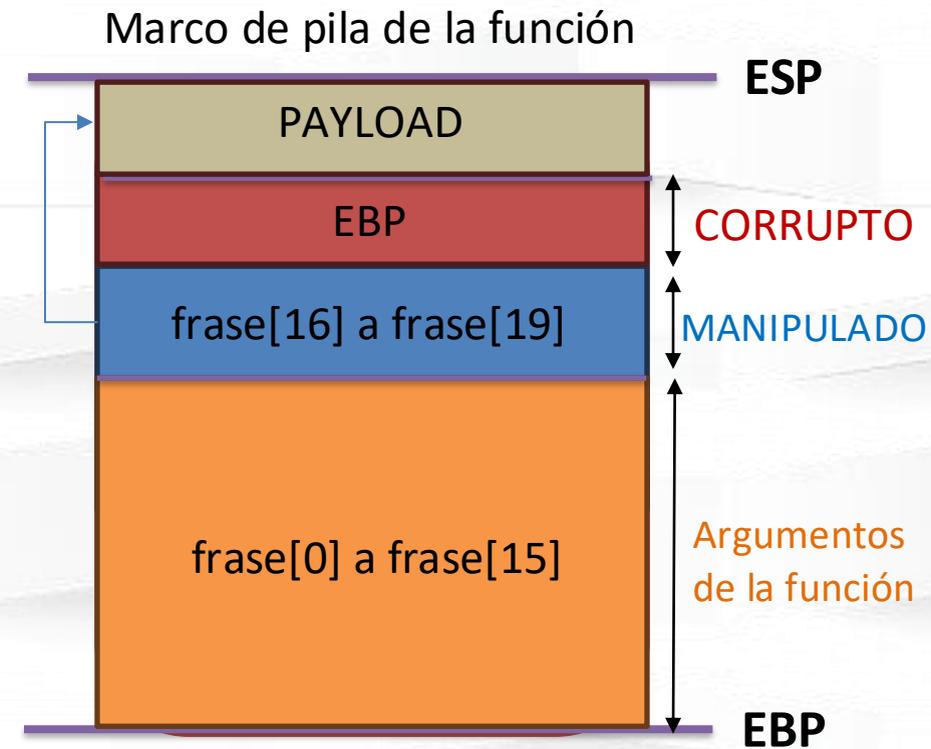
Buffer Overflow

```
void copia_cadena(char *cadena) {  
    char buffer[16];  
    strcpy(buffer, cadena);  
}  
  
void main() {  
    char frase[256];  
    int i;  
    for(i = 0; i < 255; i++) {  
        frase[i] = 'M';  
    }  
    copia_cadena(frase);  
}
```



Buffer Overflow

```
void copia_cadena(char *cadena) {  
    char buffer[16];  
    strcpy(buffer, cadena);  
}  
  
void main() {  
    char frase[16];  
    printf("Introduzca una frase");  
    scanf("%s", &frase);  
    copia_cadena(frase);  
}
```



Buffer overflow

- Partiendo de una vulnerabilidad de buffer overflow se puede llegar a aplicar técnicas interesantes:
 - ROP
 - ret2win
 - ret2libc
- Veamos algunos ejemplos

#CátedrasCiber

Módulo IV: Ingeniería inversa y Explotación de binarios