Análisis de malware

Anti-desensamblado

Gustavo Romero López - gustavo@ugr.es

Updated: 3 de marzo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Por donde vamos...

Parte 1: Análisis básico

Capítulo 3: Análisis dinámico básico

Parte 2: Análisis estático avanzado

Capítulo 7: Análisis de programas maliciosos para Windows

Parte 3: Análisis dinámico avanzado

Capítulo 8: Depuración

Capítulo 9: OllyDbg

Parte 4: Funcionalidad del malware

Capítulo 12: Ejecución encubierta

Parte 5: Anti-ingeniería inversa

Capítulo 15: Anti-desensamblado

Capítulo 16: Anti-depuración

Capítulo 17: Anti-máquinas virtuales

Capítulo 18: Compresores y descompresión

Parte 6: Temas especiales

Anti-desensamblado

1. ¿Qué es el anti-desensamblado?

2. ¿Cómo derrotar a los algoritmos de desensamblado?

3. Técnicas anti-desensamblado

4. Oscureciendo el control de flujo

5. Frustrando el análisis del marco de pila

¿Qué es el anti-desensamblado?

- Código o datos especialmente diseñados para dar lugar a un desensamblado incorrecto
- Desensamblado de idéntica secuencia de bytes:

```
                jmp     short near ptr loc_2+1
; ---------------------------------------------------------------------------

loc_2:
                call    near ptr 15FF2A71h ❶
                or      [ecx], dl
                inc     eax
; ---------------------------------------------------------------------------
                db    0
```

lineal :(

```
                jmp     short loc_3
; ---------------------------------------------------------------------------
                db 0E8h
; ---------------------------------------------------------------------------

loc_3:
                push    2Ah
                call    Sleep ❶
```

flujo :)

Desensamblado lineal I

- Desensambla instrucción por instrucción
- Método empleado por la mayoría de los depuradores
- Cada código de operación indica cual es la longitud de la instrucción
- Fácil de implementar y rápido de ejecutar
- Puede cometer errores incluso en código no malicioso por..
  - mezclar código y datos
  - bifurcaciones al medio de una instrucción
- Analogía: quitar los espacios en blanco: **...desola...**
  - ...de sol a sol...
  - ...desolado...
- Uso habitual: añadir bytes de instrucciones largas

Desensamblado lineal II

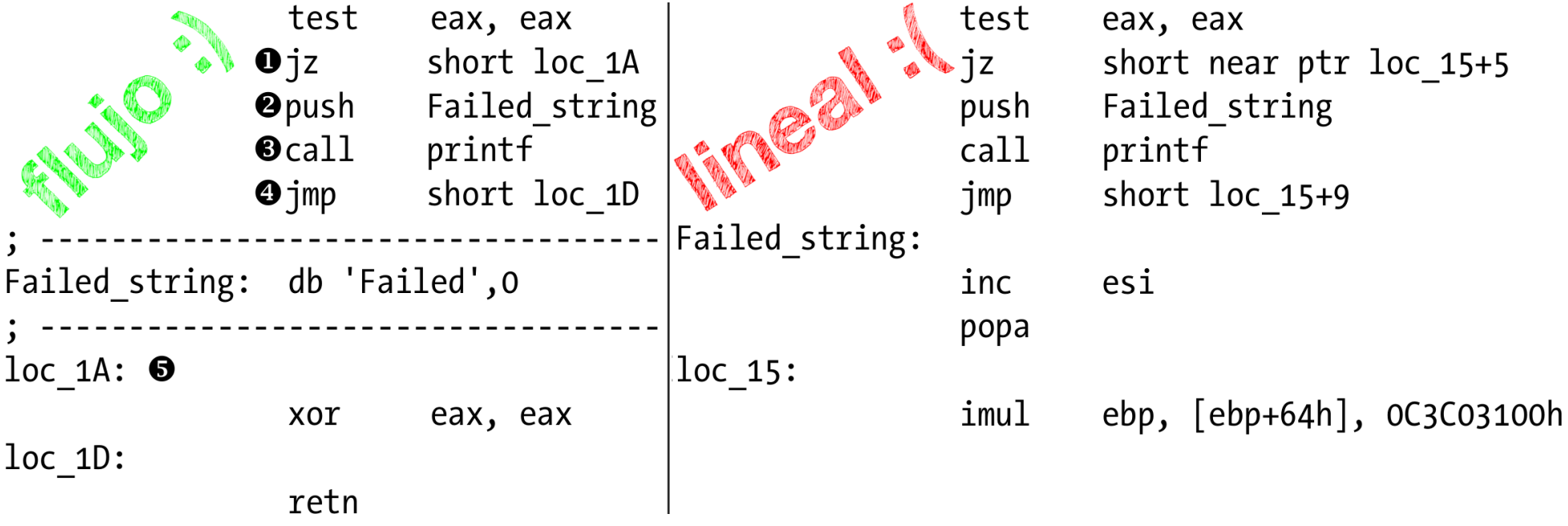
```
          jmp     ds:off_401050[eax*4] ; switch jump

          ; switch cases omitted ...

          xor     eax, eax
          pop     esi
          retn
; ---------------------------------------------------------------------------
off_401050 ❶dd offset loc_401020   ; DATA XREF: _main+19r
             dd offset loc_401027   ; jump table for switch statement
             dd offset loc_40102E
             dd offset loc_401035
```

```
and [eax],dl
inc eax
add [edi],ah
adc [eax+0x0],al
adc cs:[eax+0x0],al
xor eax,0x4010
```

byte sequence 20 10 40 00

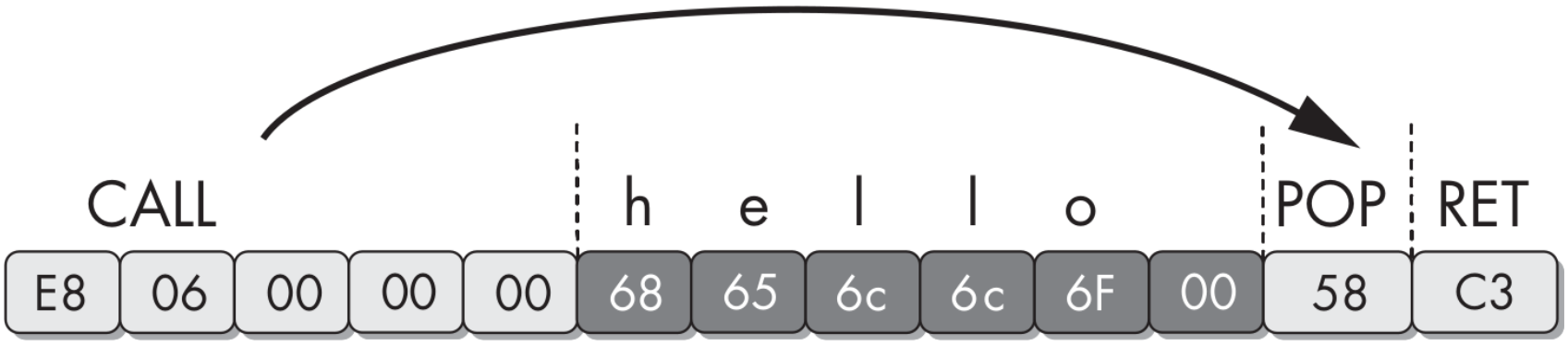
Desensamblado orientado a flujo

- Método empleado en herramientas de ingeniería inversa
- Sólo instrucciones a las que llega el control de flujo
- Comienza por las direcciones a las que no se salta
- Considera como datos al resto de la información



Desensamblado orientado a flujo (2)

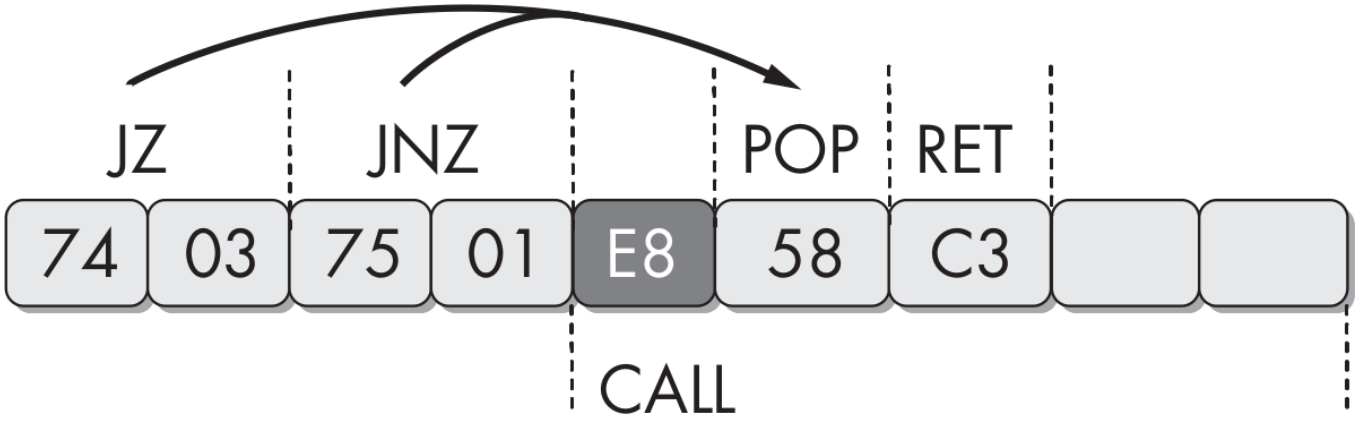
- Algunas veces incluso este método puede fallar
- Se puede forzar el cambio código/datos (C/D)




```
E8 06 00 00 00        call    near ptr loc_4011CA+1
68 65 6C 6C 6F      ❶push     6F6C6C65h

                    loc_4011CA:
00 58 C3            add       [eax-3Dh], bl
```

```
E8 06 00 00 00                      call    loc_4011CB
68 65 6C 6C 6F 00   aHello          db 'hello',0
                                    loc_4011CB:
58                                  pop     eax
C3                                  retn
```

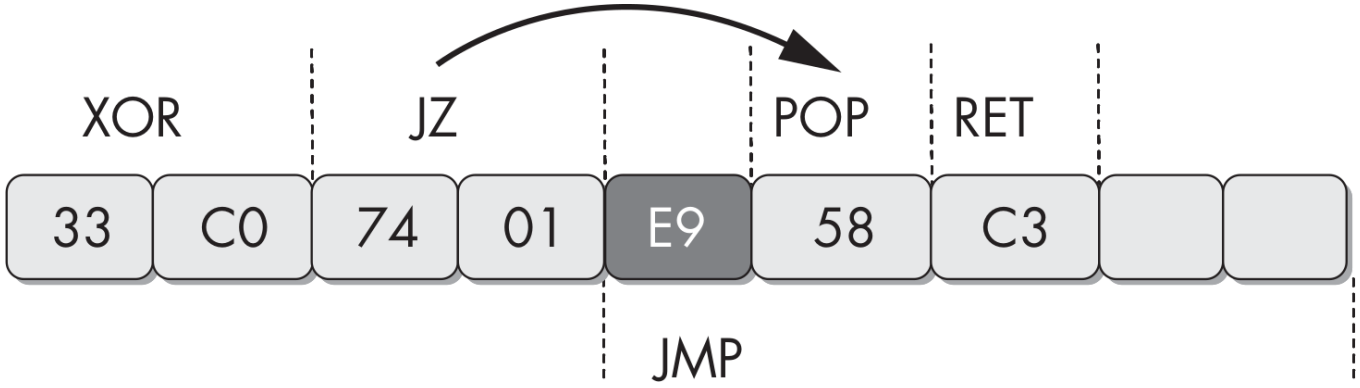
Instrucciones de salto con el mismo destino




```
74 03                   jz      short near ptr loc_4011C4+1
75 01                   jnz     short near ptr loc_4011C4+1
                        loc_4011C4:                         ; CODE XREF: sub_4011C0
                                                            ; ❷sub_4011C0+2j
E8 58 C3 90 90          ❶call    near ptr 90D0D521h
```

```
74 03                   jz      short near ptr loc_4011C5
75 01                   jnz     short near ptr loc_4011C5
        ; ---------------------------------------------------------------------
E8                      db 0E8h
        ; ---------------------------------------------------------------------
                        loc_4011C5:                         ; CODE XREF: sub_4011C0
                                                            ; sub_4011C0+2j
58                      pop     eax
C3                      retn
```

Instrucciones de salto con una condición constante

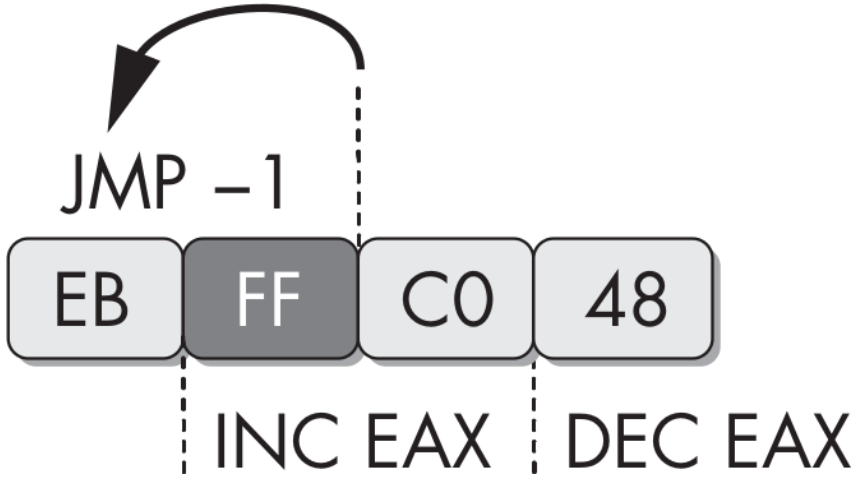



```
33 C0                   xor     eax, eax
74 01                   jz      short near ptr loc_4011C4+1
        loc_4011C4:                                 ; CODE XREF: 004011C2j
                                                    ; DATA XREF: .rdata:004020ACo
E9 58 C3 68 94          jmp     near ptr 94A8D521h
```

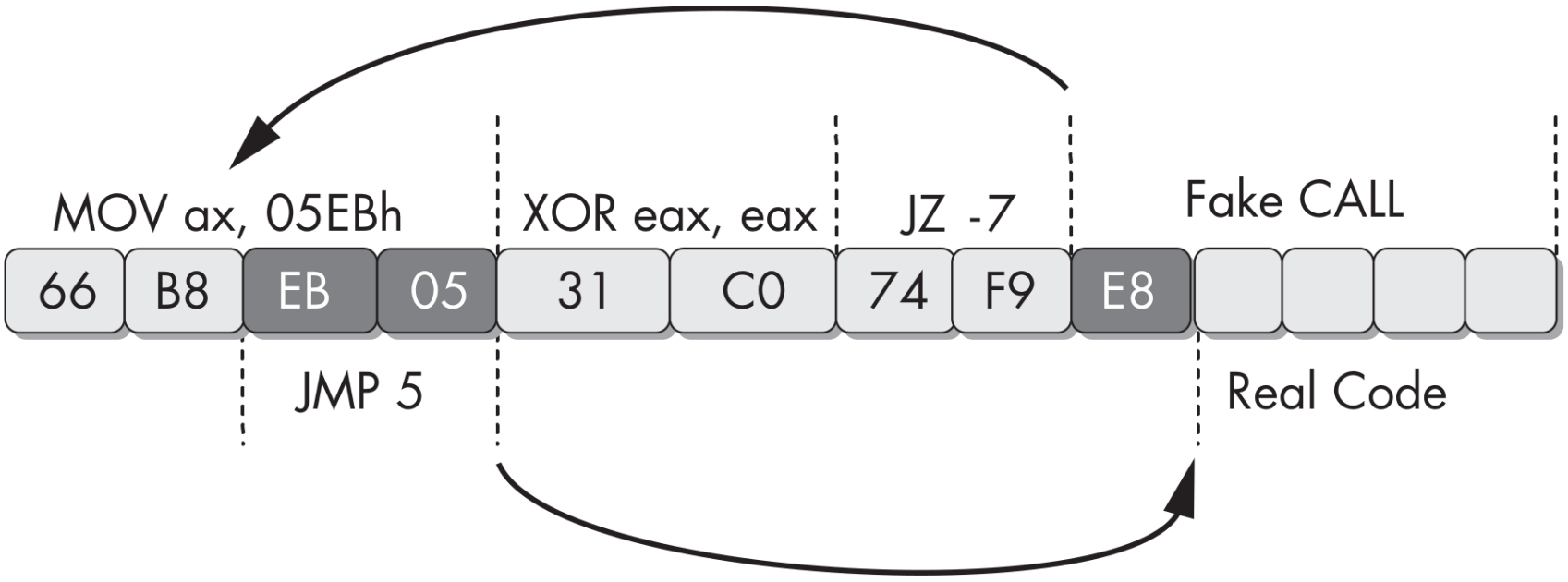
```
33 C0                   xor     eax, eax
74 01                   jz      short near ptr loc_4011C5
        ; ---------------------------------------------------------------------
E9                      db 0E9h
        ; ---------------------------------------------------------------------
        loc_4011C5:                                 ; CODE XREF: 004011C2j
                                                    ; DATA XREF: .rdata:004020ACo
58                      pop     eax
C3                      retn
```

Desensamblado imposible

- "**byte pícaro**"
  - se añade para engañar
  - puede ser ignorado
- ¿Pero y si no pudiera ser ignorado por formar parte de una o más instrucciones?
- Ningún desensamblador puede producir código adecuado
- El procesador no tiene ningún problema en ejecutarlo
- "Apaños": sustituir por NOPs o cambiar tipo a datos



Desensamblado imposible (2)




```
66 B8 EB 05             mov     ax, 5EBh
31 C0                   xor     eax, eax
74 F9                   jz      short near ptr sub_4011C0+1
                loc_4011C8:
E8 58 C3 90 90          call    near ptr 98A8D525h
```

Desensamblado imposible (3)

```
66                      byte_4011C0     db 66h
B8                                      db 0B8h
EB                                      db 0EBh
05                                      db    5
                        ; ---------------------------------------------------------
31 C0                                   xor     eax, eax
                        ; ---------------------------------------------------------
74                                      db 74h
F9                                      db 0F9h
E8                                      db 0E8h
                        ; ---------------------------------------------------------
58                                      pop     eax
C3                                      retn
```

```
90                          nop
90                          nop
90                          nop
90                          nop
31 C0                       xor     eax, eax
90                          nop
90                          nop
90                          nop
58                          pop     eax
C3                          retn
```

El problema de los punteros a función

- Los punteros a función se usan en muchos lenguajes de programción
- Utilizarlos de formar no estándar entorpece el análisis
- Se mostrarán llamadas sin saber conectarlas a la función llamada
- Podemos añadir la referencia cruzada así:
  - `AddCodeXref(0x4011DE, 0x4011C0, fl_CF)`
  - `AddCodeXref(0x4011EA, 0x4011C0, fl_CF)`
- En siguiente ejemplo se llama **dos** veces a la función `sub_4011C0` pero sólo se muestra **una** referencia

El problema de los punteros a función (2)

```
004011D0 sub_4011D0      proc near               ; CODE XREF: _main+19p
004011D0                                         ; sub_401040+8Bp
004011D0 var_4           = dword ptr -4
004011D0 arg_0           = dword ptr  8
004011D0
004011D0                 push    ebp
004011D1                 mov     ebp, esp
004011D3                 push    ecx
004011D4                 push    esi
004011D5                 mov   ❶[ebp+var_4], offset sub_4011C0
004011DC                 push    2Ah
004011DE                 call  ❷[ebp+var_4]
004011E1                 add     esp, 4
004011E4                 mov     esi, eax
004011E6                 mov     eax, [ebp+arg_0]
004011E9                 push    eax
004011EA                 call  ❸[ebp+var_4]
004011ED                 add     esp, 4
004011F0                 lea     eax, [esi+eax+1]
004011F4                 pop     esi
004011F5                 mov     esp, ebp
004011F7                 pop     ebp
004011F8                 retn
004011F8 sub_4011D0      endp
```

Abuso del punto de retorno

- `call f` ≔ "`push %eip`" + "`jmp f`"
- El punto de retorno será la siguiente dirección de memoria tras el `call`
- `ret` ≔ "`pop %eip`" + "`jmp %eip`"
- Efectos:
  - no se muestran las referencias cruzadas con la función
  - se deja de desensamblar la función prematuramente
- Reparación: cambiar por `NOP`s

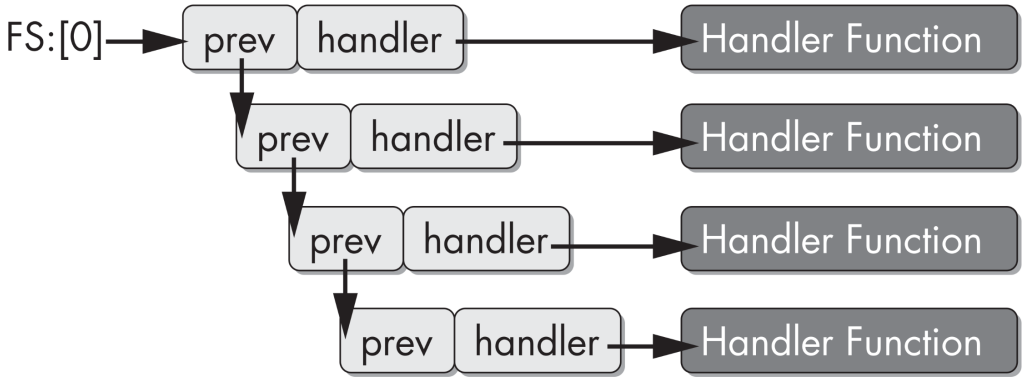
Abuso del punto de retorno (2)

```
004011C0 sub_4011C0      proc near               ; CODE XREF: _main+19p
004011C0                                         ; sub_401040+8Bp
004011C0
004011C0 var_4           = byte ptr -4
004011C0
004011C0                 call    $+5
004011C5                 add     [esp+4+var_4], 5
004011C9                 retn
004011C9 sub_4011C0      endp ; sp-analysis failed
004011C9
004011CA ; ---------------------------------------------------------------
004011CA                 push    ebp
004011CB                 mov     ebp, esp
004011CD                 mov     eax, [ebp+8]
004011D0                 imul    eax, 2Ah
004011D3                 mov     esp, ebp
004011D5                 pop     ebp
004011D6                 retn
```

Abuso de manejador de excepción estructurado “SEH”

- Otro mecanismo para enturbiar el control de flujo.
- Lista enlazada de punteros a función en pila:
- `fs –> TEB –> TIB –> SEH –>`

```
struct _EXCEPTION_REGISTRATION { DWORD prev, handler };
```



- Uso típico: añadirse al pricipio de la lista

```
push ExceptionHandler
push fs:[0]
mov  fs:[0], esp
```

- Los desensambladores desconocen que código ejecutar y no pueden analizarlo.

Abuso de manejador de excepción estructurado (2)

```
00401050                ❷mov     eax, (offset loc_40106B+1)
00401055                  add     eax, 14h
00401058                  push    eax
00401059                  push    large dword ptr fs:0 ; dwMilliseconds
00401060                  mov     large fs:0, esp
00401067                  xor     ecx, ecx
00401069                ❸div     ecx
0040106B
0040106B loc_40106B:                                ; DATA XREF: sub_401050o
0040106B                  call    near ptr Sleep
00401070                  retn
00401070 sub_401050       endp ; sp-analysis failed
00401070
00401070 ; ---------------------------------------------------------------------
00401071                  align 10h
00401080                ❶dd 824648Bh, 0A164h, 8B0000h, 0A364008Bh, 0
00401094                  dd 6808C483h
00401098                  dd offset aMysteryCode  ; "Mystery Code"
0040109C                  dd 2DE8h, 4C4830Oh, 3 dup(0CCCCCCCCh)
```

Abuso de manejador de excepción estructurado (3)

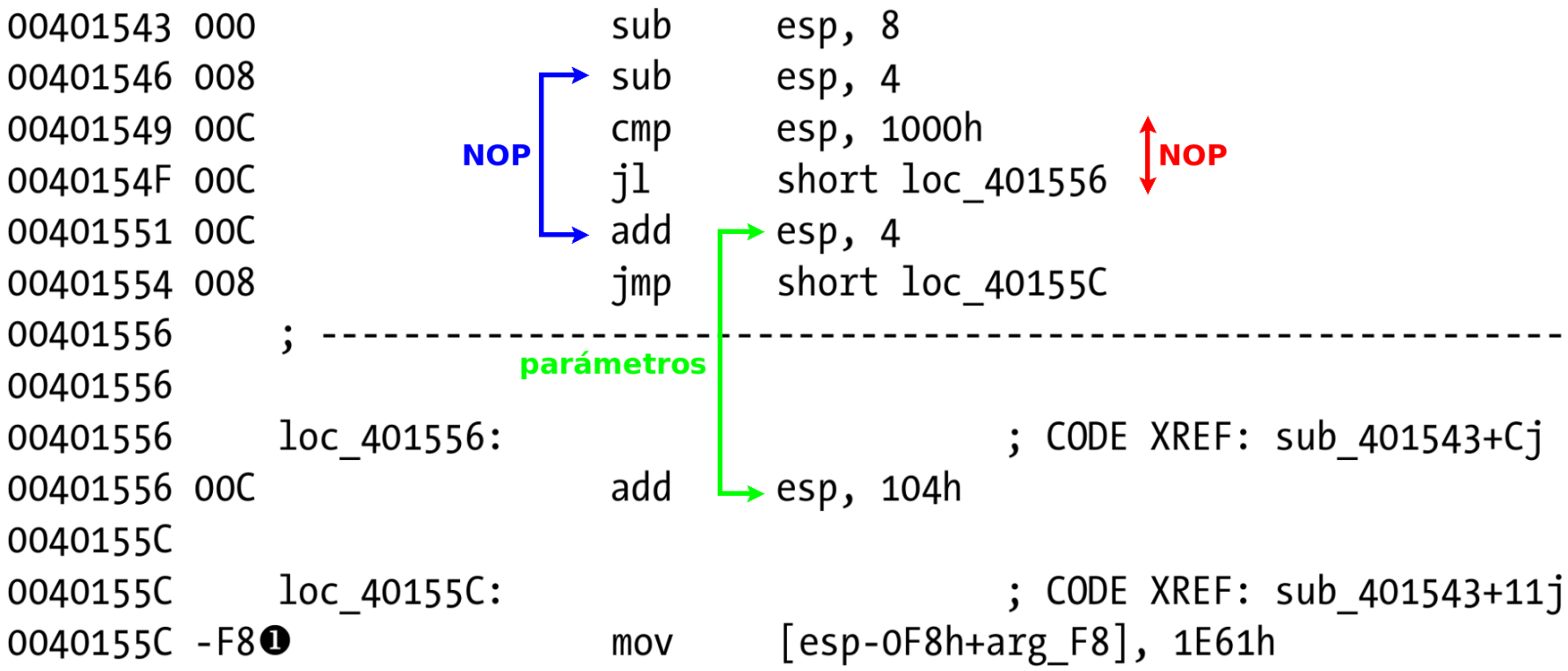
- Usando la tecla C en IDA podemos convertir en código
- Funcionalidad:
  - elimina la función de la cadena SEH
  - imprime un mensaje

```
00401080        mov     esp, [esp+8]
00401084        mov     eax, large fs:0
0040108A        mov     eax, [eax]
0040108C        mov     eax, [eax]
0040108E        mov     large fs:0, eax
00401094        add     esp, 8
00401097        push    offset aMysteryCode ; "Mystery Code"
0040109C        call    printf
```

Frustrando el análisis del marco de pila

```
00401543     sub_401543      proc near           ; CODE XREF: sub_4012D0+3Cp
00401543                                         ; sub_401328+9Bp
00401543
00401543     arg_F4          = dword ptr  0F8h
00401543     arg_F8          = dword ptr  0FCh
00401543
00401543 000                 sub     esp, 8
00401546 008                 sub     esp, 4
00401549 00C                 cmp     esp, 1000h
0040154F 00C                 jl      short loc_401556
00401551 00C                 add     esp, 4
00401554 008                 jmp     short loc_40155C
00401556     ; ---------------------------------------------------------------
00401556
00401556     loc_401556:                         ; CODE XREF: sub_401543+Cj
00401556 00C                 add     esp, 104h
0040155C
0040155C     loc_40155C:                         ; CODE XREF: sub_401543+11j
0040155C -F8❶                mov     [esp-0F8h+arg_F8], 1E61h
00401564 -F8                 lea     eax, [esp-0F8h+arg_F8]
00401568 -F8                 mov     [esp-0F8h+arg_F4], eax
0040156B -F8                 mov     edx, [esp-0F8h+arg_F4]
0040156E -F8                 mov     eax, [esp-0F8h+arg_F8]
00401572 -F8                 inc     eax
00401573 -F8                 mov     [edx], eax
00401575 -F8                 mov     eax, [esp-0F8h+arg_F4]
00401578 -F8                 mov     eax, [eax]
0040157A -F8                 add     esp, 8
0040157D -100                retn
0040157D     sub_401543      endp ; sp-analysis failed
```

Frustrando el análisis del marco de pila (2)



Ejercicios: Lab 15-1

Analice el programa Lab15-01.exe. Imprime el mensaje “Good Job!” al pasarle el parámetro adecuado a través de la línea de órdenes.

1. ¿Qué técnicas emplea para evitar el desensamblado?
2. ¿Qué códigos de operación usa para estropear el desensamblado?
3. ¿Cuántas veces se utiliza estas ténicas?
4. ¿Qué parámetro hará que se muestre el mensaje “Good Job!”?

Ejercicios: Lab 15-2

Analice el programa Lab15-02.exe. Corrija todas la contramedidas para evitar el desensamblado antes de analizar el binario.

1. ¿Qué URL solicita el programa?
2. ¿Cómo se genera la cadena del "agente de usuario"?
3. ¿Qué busca el programa en la página solicitada al inicio?
4. ¿Qué hace el programa con la información que extrae de la página web?