



Retomando

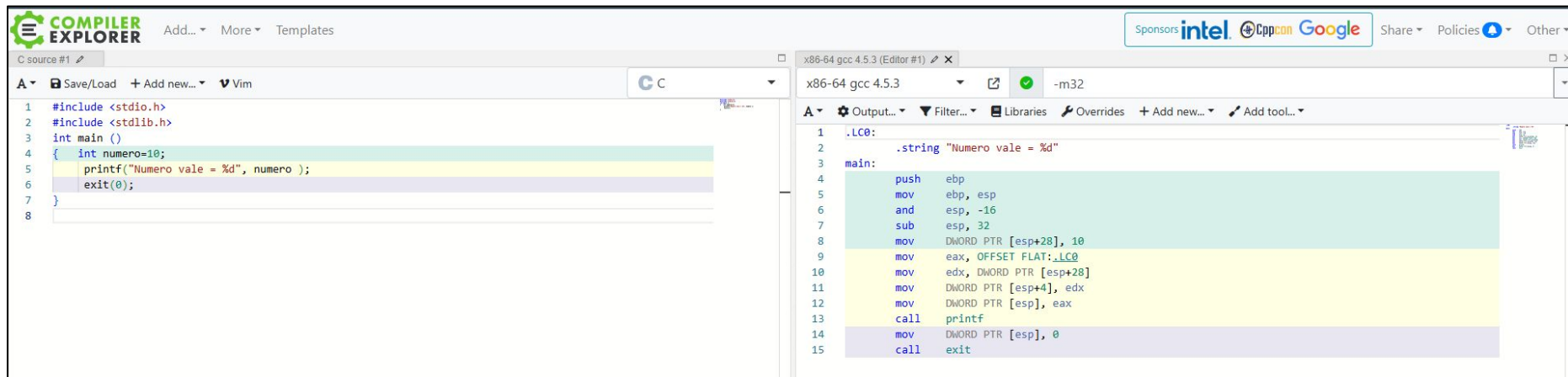
Ejemplo anterior

```
/* asmyc1.c */  
#include <stdio.h>  
#include <stdlib.h>  
  
int main ()  
{ int numero=10;  
  printf("Numero vale = %d", numero );  
  exit(0);  
}
```

Podemos cargar este código en C en el sitio: <https://godbolt.org/>
Elegir la versión de compilador y ver su salida en ASM

Cada línea de C muestra las n líneas de ASM

<https://godbolt.org/z/o1qaG1v6G>



The screenshot displays the Compiler Explorer interface. On the left, the C source code is shown in a file named 'C source #1'. The code is as follows:

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 int main ()  
4 { int numero=10;  
5   printf("Numero vale = %d", numero );  
6   exit(0);  
7 }  
8
```

On the right, the assembly output for 'x86-64 gcc 4.5.3' is displayed. The assembly code is as follows:

```
1 .LC0:  
2   .string "Numero vale = %d"  
3 main:  
4   push    ebp  
5   mov     ebp, esp  
6   and     esp, -16  
7   sub     esp, 32  
8   mov     DWORD PTR [esp+28], 10  
9   mov     eax, OFFSET FLAT:.LC0  
10  mov     edx, DWORD PTR [esp+28]  
11  mov     DWORD PTR [esp+4], edx  
12  mov     DWORD PTR [esp], eax  
13  call     printf  
14  mov     DWORD PTR [esp], 0  
15  call     exit
```

Ejemplo de Salida en ASM 64 bits

Es el mismo ejercicio que la clase pasada vimos en 32 bits.

```
.file    "asmyc.c"
.section .rodata
.LC0:
.string  "numero vale = %d"
.text
.global  main
.type    main, @function
main:
.LFB2:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 16
```

```
    mov     DWORD PTR [rbp-4], 10
    mov     eax, DWORD PTR [rbp-4]
    mov     esi, eax
    mov     edi, OFFSET FLAT:.LC0
    call    printf
    mov     edi, 0
    call    exit
.LFE2:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 4.8.4)"

.section .note.GNU-stack,"",@progbits
```



Análisis de manejo de pila

Análisis detallado de manejo de pila

//detalle1.c

// Programa para analizar en detalle los stack frame

```
int suma( int sum1, int sum2)
```

```
{
```

```
    return sum1+sum2;
```

```
}
```

```
int main(void)
```

```
{
```

```
    suma(3,4);
```

```
    return 0;
```

```
}
```

Análisis detallado de manejo de pila

(gdb) set disassembly-flavor intel

(gdb) disassemble main

Dump of assembler code for function main:

0x080483e9 <+0>: push ebp

0x080483ea <+1>: mov ebp,esp

0x080483ec <+3>: sub esp,0x8

0x080483ef <+6>: mov DWORD PTR [esp+0x4],0x4

0x080483f7 <+14>: mov DWORD PTR [esp],0x3

0x080483fe <+21>: call 0x80483dc <suma>

0x08048403 <+26>: mov eax,0x0

0x08048408 <+31>: leave

0x08048409 <+32>: ret

End of assembler dump.

Análisis detallado de manejo de pila

(gdb) disassemble suma

Dump of assembler code for function suma:

```
0x080483dc <+0>:  push  ebp
0x080483dd <+1>:  mov   ebp,esp
0x080483df <+3>:  mov   eax,DWORD PTR [ebp+0xc]
0x080483e2 <+6>:  mov   edx,DWORD PTR [ebp+0x8]
0x080483e5 <+9>:  add   eax,edx
0x080483e7 <+11>: pop   ebp
0x080483e8 <+12>:  ret
```

End of assembler dump.

(gdb)

Análisis – Ejemplo 2

Repetiremos el análisis pero agregando una variable local a la función main().

```
//detalle2.c
```

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

```
int suma( int sum1, int sum2)
```

```
{
```

```
    return sum1+sum2;
```

```
}
```

```
int main(void)
```

```
{
```

```
    int resul;
```

```
    resul=suma(3,4);
```

```
    return 0;
```

```
}
```


Análisis – Ejemplo 2

Dump of assembler code for function main:

```
0x080483e9 <+0>:  push  ebp  
0x080483ea <+1>:  mov   ebp,esp  
0x080483ec <+3>:  sub   esp,0x18  
0x080483ef <+6>:  mov   DWORD PTR [esp+0x4],0x4  
0x080483f7 <+14>:  mov   DWORD PTR [esp],0x3  
0x080483fe <+21>:  call  0x80483dc <suma>  
0x08048403 <+26>:  mov   DWORD PTR [ebp-0x4],eax  
0x08048406 <+29>:  mov   eax,0x0  
0x0804840b <+34>:  leave  
0x0804840c <+35>:  ret
```

End of assembler dump.

Análisis de pila – Ejemplo 3

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

int main(void)
{
    int pass = 0;
    char buff[15];

    printf("\n Enter the password : \n");
    gets(buff);

    if(strcmp(buff, "thegeekstuff")!=0)
    {
        printf ("\n Wrong Password \n");
    }
}
```

```
else
{
    printf ("\n Correct Password \n");
    pass = 1;
}

if(pass!=0)
{
    /* Now Give root or admin rights to user*/
    printf ("\n Root privileges given to the user
\n");
}

return 0;
}
```

Análisis Ejemplo 2

Al compilar en forma clásica

```
gcc detalle3.c -o detalle3
```

```
[svalles@pampero teoria]$ ./detalle3
```

```
Enter the password :
thesecretpass
```

Wrong Password

```
[svalles@pampero teoria]$
```

```
[svalles@pampero teoria]$ ./detalle3
```

Enter the password :

```
a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a a j j j j j j j j j j j j j j j j j j j j j j j j j j j
```

Wrong Password

```
*** stack smashing detected ***: <unknown> terminated
```

Aborted (core dumped)

```
[svalles@pampero teoria]$
```

Análisis – Ejemplo 3

Al compilar sin protección de stack

```
gcc detalle3.c -o detalle3 -fno-stack-protector
```

```
[svalles@pampero teoria]$ ./detalle3sinprotect

Enter the password :
123456789012345

Wrong Password
[svalles@pampero teoria]$ ./detalle3sinprotect

Enter the password :
1234567890123451

Wrong Password

Root privileges given to the user
[svalles@pampero teoria]$
```

Salida en ASM de Ejemplo 3

```
.file "detalle3.c"
.intel_syntax noprefix
.text
.section      .rodata
.LC0:
.string "\n Enter the password : "
.LC1:
.string "thegeekstuff"
.LC2:
.string "\n Wrong Password "
.LC3:
.string "\n Correct Password "
.align 8
.LC4:
.string "\n Root privileges given to
the user "
.text
.globl main
.type  main, @function
```

```
main:
.LFB0:
.cfi_startproc
push    rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
mov     rbp, rsp
.cfi_def_cfa_register 6
sub     rsp, 32
mov     rax, QWORD PTR fs:40
mov     QWORD PTR -8[rbp], rax
xor     eax, eax
mov     DWORD PTR -28[rbp], 0
lea     rdi, .LC0[rip]
call    puts@PLT
lea     rax, -23[rbp]
mov     rdi, rax
mov     eax, 0
```

Salida en ASM de Ejemplo 3

```
call  gets@PLT
    lea  rax, -23[rbp]
    lea  rsi, .LC1[rip]
    mov  rdi, rax
    call strcmp@PLT
    test eax, eax
    je   .L2
    lea  rdi, .LC2[rip]
    call puts@PLT
    jmp  .L3
.L2:
    lea  rdi, .LC3[rip]
    call puts@PLT
    mov  DWORD PTR -28[rbp], 1
.L3:
    cmp  DWORD PTR -28[rbp], 0
    je   .L4
    lea  rdi, .LC4[rip]
    call puts@PLT
```

```
.L4:
    mov  eax, 0
    mov  rdx, QWORD PTR -8[rbp]
    xor  rdx, QWORD PTR fs:40
    je   .L6
    call  __stack_chk_fail@PLT
.L6:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size  main, .-main
    .ident "GCC: (GNU) 7.3.0"
    .section      .note.GNU-stack,"",@progbits
```

SPP (Stack Smashing Protector)

- Lo implementa gcc
- Utiliza la función `__stack_chk_fail`
- Ubica un valor entre EBP y las variables locales que se lo denomina **CANARY**
- Antes de retornar verifica que el CANARY no haya sido modificado
- Si lo fue termina la ejecución.



SPP (Stack Smashing Protector)

- Lo implementa gcc
- Utiliza la función `__stack_chk_fail`
- Ubica un valor entre EBP y las variables locales que se lo denomina **CANARY**
- Antes de retornar verifica que el CANARY no haya sido modificado
- Si lo fue termina la ejecución.

