

Memoria sobre la primera fase de HackingForce

Grupo 4 – PL4, FCR GIITIN01 (20-21)

UO283319 – Juan Francisco Mier Montoto

UO282574 - Miguel del Riego Lázaro

UO281892 - Ignacio Valdés Azorín

Índice

- [Fase 1.1 – Ceremonia de iniciación](#)
 - [Explicación del código](#)
 - [Ejemplos de entradas válidas e inválidas](#)
- [Fase 1.2 – Supervivencia en código máquina](#)
 - [Introducción general](#)
 - [Pregunta 1](#)
 - [Pregunta 2](#)
 - [Pregunta 3](#)
 - [Pregunta 4](#)
- [Reparto de trabajo](#)

Fase 1.1 – Ceremonia de iniciación

Explicación del código

El código está compuesto de tres partes fundamentales:

- El receptor de errores, que recibe la cadena de error correspondiente después de que se haya incumplido una de las condiciones, la muestra por pantalla y sale del programa con ERRORLEVEL establecido a 1.
- Los métodos principales del programa, que desarrollan las tareas especificadas, incluyendo las funciones de Assembly.
- El main, que se encarga de dirigir el programa y mostrar el mensaje de confirmación en caso de que no se salga antes del problema.

Ejemplos de entradas válidas e inválidas

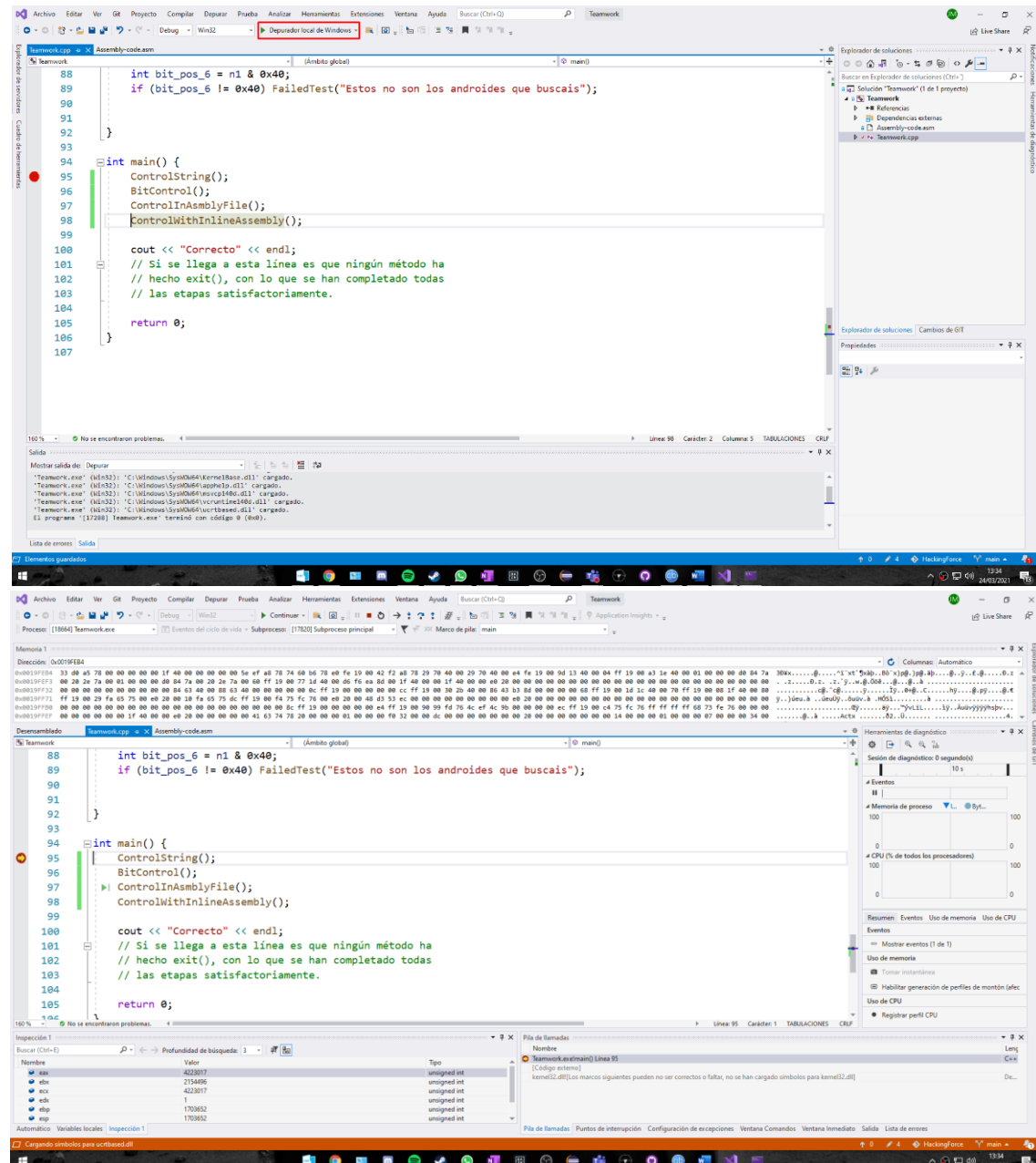
Para cada método, hay una combinación infinita de entradas válidas e inválidas. Estos son algunos ejemplos de ambos casos:

- **ControlString():**
 - En el caso de la primera cadena, solo es válida la entrada “otîPqhXcD”. Cualquier otra entrada será inválida y devolverá la cadena “Sayonara, baby” al receptor de errores.
 - En el caso de la segunda cadena, para que sea válida debe tener una longitud de al menos 6 y que el carácter de índice 3 sea igual al de índice 5, como por ejemplo “BBBaBaBB” o “f8ds0sa2d02”. Cualquier otra entrada que no cumpla estas condiciones será inválida, como por ejemplo “abcd” o “123456789”.
- **BitControl():**
 - En la primera condición, los bits entre los índices 2 y 4 (ambos incluidos) del segundo número introducido deben de ser iguales a la representación en binario de 1, es decir, 001. Algunos ejemplos de entradas válidas del segundo número son “740” o “4”.
 - Para la segunda etapa, se aplican máscaras a ambos números y se combinan los resultados, de manera que el nuevo número tiene los 23 bits de mayor peso del primer número y los 9 bits restantes de menor peso del segundo. Además, el bit 6 del primer número debe de ser 1. Utilizando respectivamente los ejemplos de entradas válidas anteriores, para 740 una entrada válida sería “22080” y para 4 una entrada válida sería “3648”.
- **ControlInAsmblyFile():**
 - Para el primer número, el valor de los 8 bits de menor peso interpretados en binario debe de ser 238, como por ejemplo 240 o 61695.
 - Para los otros dos números, sus bits de la posición 8 deben coincidir, como por ejemplo 256 y 16777472 o 768 y 260.
- **ControlWithInlineAssembly():**
 - Los 16 bits de mayor peso del número introducido deben de ser iguales que los 16 bits restantes, como por ejemplo el 0 o el 143198345.

Fase 1.2 – Salvando al mundo

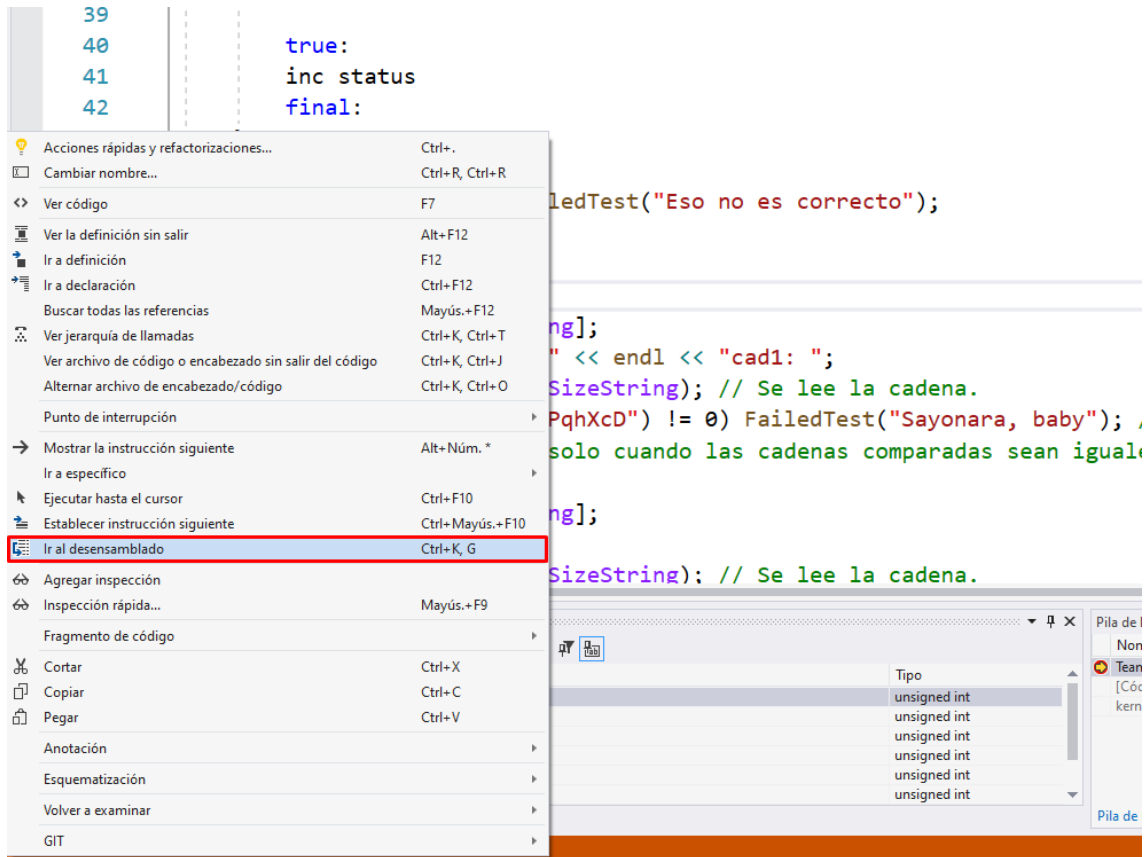
Introducción general

En este apartado se responderán a las preguntas propuestas en el enunciado del trabajo, incluyendo capturas de pantalla de los resultados. Las respuestas se consiguen mediante el depurador de Visual Studio en combinación con otras herramientas como la Memoria en ventana y el modo desensamblado. Para entrar en el modo depuración, primero se inserta un punto de interrupción y luego se extrae la información necesaria en el modo “Debug”.

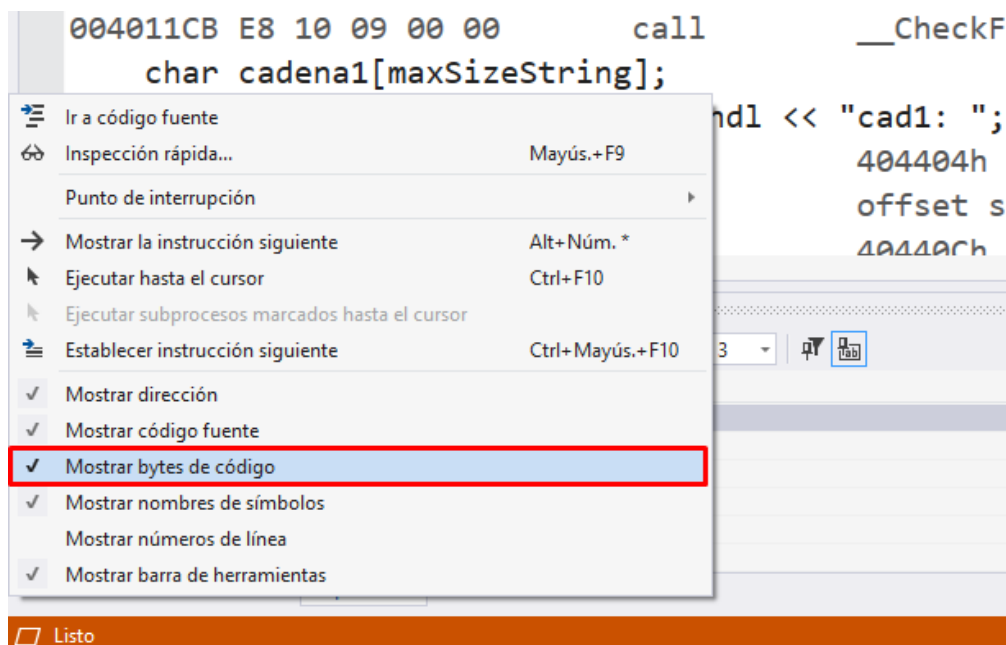


Pregunta 1.

Para acceder al desensamblado, se introduce un punto de interrupción y se hace clic derecho sobre la instrucción que queremos observar en desensamblado. En el menú desplegable, se selecciona "Ir al desensamblado":



Una vez en el desensamblado del código, para poder ver el código máquina debemos activar la siguiente opción, accesible igual que la anterior mediante el clic derecho:



La primera dirección de memoria del paso de parámetros a la función en Assembly “IsValidAssembly(int, int, int)” es “004010DB”, resaltado en la imagen en rojo. El mnemónico de esta instrucción es “mov ecx, [...] num3”, la parte resaltada en verde. Por último, en azul, se encuentra el código máquina en hexadecimal. Los pasos de parámetros continúan hasta la dirección “004010E7”, donde se llama al procedimiento.

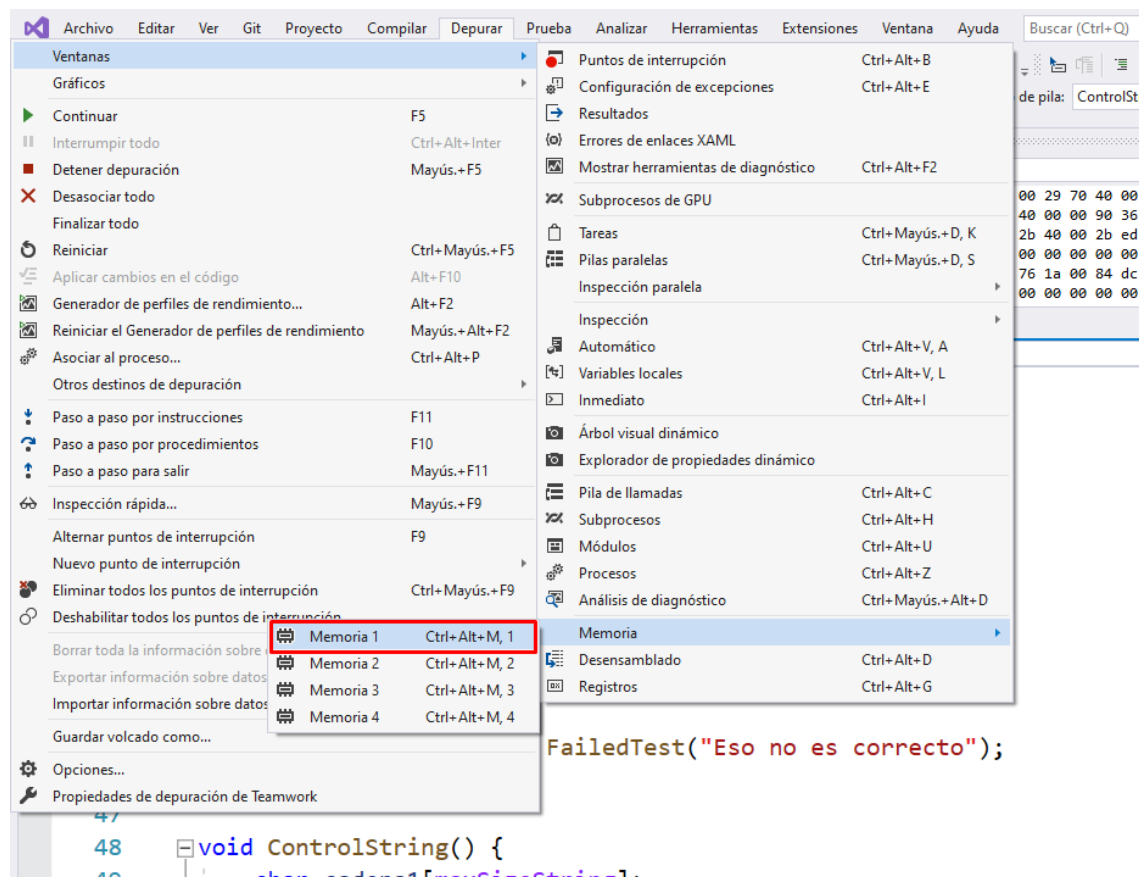
```

    if (!IsValidAssembly(num1, num2, num3)) FailedTest("Eso no es correcto");
004010DB 8B 4D FC      mov     ecx,dword ptr [num3]
004010DE 51           push    ecx
004010DF 8B 55 F8      mov     edx,dword ptr [num2]
004010E2 52           push    edx
004010E3 8B 45 F4      mov     eax,dword ptr [num1]
004010E6 50           push    eax
004010E7 E8 3C 09 00 00 call    IsValidAssembly (00401A28h)

```

Pregunta 2.

Para obtener la dirección de memoria en la que se encuentra la primera cadena de “ControlString()”, tenemos que acceder a la ventana de “Memoria 1” mientras se ejecuta la depuración:



Ahora, solo hay que introducir el nombre de la cadena, en este caso “cadena1”. Así se puede observar que dicha dirección es “0x0019FEB4”.

The screenshot shows a debugger window with the 'Memoria 1' tab selected. The 'Dirección:' field contains 'cadena1'. Below it, a memory dump shows the address 0x0019FEB4 highlighted in red, with the value 33 d0 a5 78 00 00 00 00 f0 1e 40 00 00 00 00 00 5e ef a8 78 74 60 b6 78 e0 fe 19 00 42 f2 a8 78. The 'Desensamblado' tab is also open, showing assembly code for 'Teamwork.cpp'. The code is as follows:

```

34      and ebx, 0x0000FFFF
35      shr eax, 16
36      cmp eax, ebx
37      je true
38      jmp final
39
40      true:
41      inc status

```

Pregunta 3.

Para obtener la dirección en memoria, el código fuente y el mnemónico del epílogo, se sigue el mismo procedimiento que en la primera pregunta. En el caso de la primera instrucción, su dirección en memoria será “00401297”, su código fuente en hexadecimal es “8BE5” y el mnemónico correspondiente es “mov esp, ebp”.

The screenshot shows a list of assembly instructions. The instruction at address 00401297 is highlighted with a red box, showing the hex code 8B E5. The instruction at address 00401299 is highlighted with a blue box, showing the hex code 5D. The instruction at address 0040129A is highlighted with a green box, showing the hex code C3. The instructions are as follows:

```

0040127E 6B C8 05      imul     ecx,eax,5
00401281 0F BE 44 0D EC movsx    eax,byte ptr cadena2[ecx]
00401286 3B D0         cmp     edx,eax
00401288 74 0D         je      ControlString+0D7h (0401297h)
0040128A 68 40 44 40 00 push    404440h
0040128F E8 6C FD FF FF call    FailedTest (0401000h)
00401294 83 C4 04      add     esp,4
// Se evalúa si la cadena tiene menos de 6 de longitud o si los caracteres de la
// posición 3 y la 5 son diferentes.
}
00401297 8B E5      mov     esp,ebp
00401299 5D         pop     ebp
0040129A C3         ret

```

Pregunta 4.

Igual que en la primera pregunta, al entrar en el modo desensamblado podemos comprobar la dirección de memoria de la primera instrucción, “0040116D”, su código máquina en hexadecimal, “8B45F4”, y por último su mnemónico, “mov eax, [...] check”.

00401150	8B 0D 4C 40 40 00	mov	ecx,dword ptr [__imp_std::cin (040404C
00401156	FF 15 40 40 40 00	call	dword ptr [__imp_std::basic_istream<ch
	cout << endl;		
0040115C	68 00 17 40 00	push	offset std::endl<char,std::char_traits
00401161	8B 0D 50 40 40 00	mov	ecx,dword ptr [__imp_std::cout (040405
00401167	FF 15 44 40 40 00	call	dword ptr [__imp_std::basic_ostream<ch
	__asm {		
	mov eax, check		
0040116D	8B 45 F4	mov	eax,dword ptr [check]
	mov ebx, check		
00401170	8B 5D F4	mov	ebx,dword ptr [check]
	and eax, 0xFFFF0000		
00401173	25 00 00 FF FF	and	eax,0FFFF000h
	and ebx, 0x0000FFFF		
00401178	81 E3 FF FF 00 00	and	ebx,0FFFFh
	shr eax, 16		
0040117E	C1 E8 10	shr	eax,10h
	cmp eax, ebx		
00401181	3B C3	cmp	eax,ebx

Reparto del trabajo

El trabajo se ha repartido de esta manera:

- Miguel del Riego: BitControl()
- Juan Mier: ControlString(), ControlInAsmbyFile(), ControlWithInlineAssembly(), FailedTest() y main(), memoria.
- Ignacio Valdés: no ha participado.