COMPUTER & NETWORKS FUNDAMENTALS' TEAMWORK

Phase 1 report

Mikel Fernández Esparta UO275688
Miguel Gayol Pertierra UO277574
Jorge Joaquín Gancedo Fernández UO282161

Index

Questions

You must indicate the process to obtain each solution and include screenshots to check where the answer comes from.

Memory address where the code to pass the parameters to the function IsValidAssembly() begins, and the code itself, both as machine code and as mnemonics.

Inspección 1

Buscar (Ctrl+E) Profundidad de búsqueda: 3

Nombre	Valor	Tipo
&input1	0x0019fed0 {0x0019fee0}	int *
&input2	0x0019fed4 {0x00408029}	int *
&input3	0x0019fed8 {0x00408029}	int *

```
; Complete the procedure
IsValidAssembly PROC

mov eax, 1024                ;eax = bit 10 = 0x400h
AND edx, eax                 ;edx = eax
shr edx, 10                  ;move to the right 10 bits
mov eax, 0                   ;eax = 0
cmp edx, eax                 ;if(bit10 == 0)
jne endBad                   ;jump to the end if they are not equal
XOR ebx, ecx                 ;input2 XOR input3
mov eax, 988921              ;eax = 988921 the result of the XOR must equal this
cmp ebx, eax                 ;if(iput2 XOR input3 == 988921)
jne endBad                   ;jump to the end if they are not equal
jmp endGood                  ;jump to endGood
endBad:
ret 0                        ;returns 0
endGood:
ret 1                        ;returns 1
IsValidAssembly ENDP

END
```

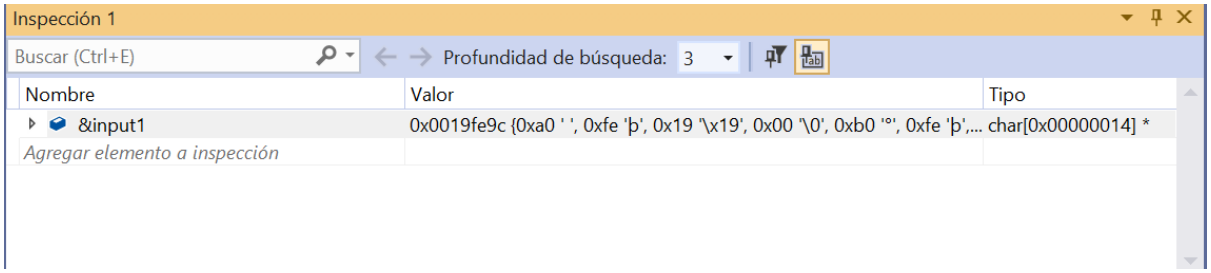
```
    int result = IsValidAssembly(input1, input2, input3);
0040131B  mov         eax,dword ptr [input3]
0040131E  push        eax
0040131F  mov         ecx,dword ptr [input2]
00401322  push        ecx
00401323  mov         edx,dword ptr [input1]
00401326  push        edx
00401327  call        IsValidAssembly (0402498h)
0040132C  add         esp,0Ch
0040132F  movzx       eax,al
00401332  mov         dword ptr [result],eax
```

```
mov eax, 1024                ;eax = bit 10 = 0x400h
00402498  mov          eax,400h
AND edx, eax                 ;edx = eax
0040249D  and          edx,eax
shr edx, 10                  ;move to the right 10 bits
0040249F  shr          edx,0Ah
mov eax, 0                   ;eax = 0
004024A2  mov          eax,0
cmp edx, eax                 ;if(bit10 == 0)
004024A7  cmp          edx,eax
jne endBad                   ;jump to the end if they are not equal
004024A9  jne          endBad (04024B8h)
XOR ebx, ecx                 ;input2 XOR input3
004024AB  xor          ebx,ecx
mov eax, 988921              ;eax = 988921 the result of the XOR must equal this
004024AD  mov          eax,0F16F9h
```

```
004024AD  mov          eax,0F16F9h
cmp ebx, eax                 ;if(iput2 XOR input3 == 988921)
004024B2  cmp          ebx,eax
jne endBad                   ;jump to the end if they are not equal
004024B4  jne          endBad (04024B8h)
jmp endGood                  ;jump to endGood
004024B6  jmp          endBad+1h (04024B9h)
endBad:
ret 0                        ;returns 0
004024B8  ret
endGood:
ret 1                        ;returns 1
004024B9  ret          1
```

Memory address where the first string read in the first function described in the instructions is stored.

When debugging (F5) after setting a breakpoint, we go to “Inspeccion 1” and add the element we want to see the memory location which is input1 in this case, but to see the location we have to put the ampersand beforehand “&input1”. Input1 is stored in the memory location 0x0019fe9c, as it can be seen in the following image.



Memory addresses where the epilogue of the first function described in the instructions is stored, and the code itself, both as machine code and as mnemonics.

We need to set a breakpoint in the main function before calling the first method “CheckPass()” go into debug mode with F10 and then right click and select (“Ir al

desensamblado”). There we can check the epilogue of the first function. We can also check the memory location where the value por input2 is stored, which is 0x0019feb0.

```
void CheckPass()
{
00401020  push        ebp
00401021  mov         ebp,esp
00401023  sub         esp,40h
00401026  mov         ecx,offset _FF17D5C0_Teamwork@cpp (0408029h)
0040102B  call        __CheckForDebuggerJustMyCode (0402810h)
    char password[maxChars] = "43WscadOB";
00401030  mov         eax,dword ptr ds:[004053D0h]
00401035  mov         dword ptr [password],eax
00401038  mov         ecx,dword ptr ds:[4053D4h]
0040103E  mov         dword ptr [ebp-14h],ecx
00401041  mov         dx,word ptr ds:[4053D8h]
00401048  mov         word ptr [ebp-10h],dx
0040104C  xor         eax,eax
0040104E  mov         dword ptr [ebp-0Eh],eax
00401051  mov         dword ptr [ebp-0Ah],eax
00401054  mov         word ptr [ebp-6],ax

    // Create the strings
    char input1[maxChars];
    char input2[maxChars];

    // Initialize the first to the input of the user
    cout << "Input the password: "; cin >> input1;
00401058  push        4053DCh
0040105D  mov         ecx,dword ptr [__imp_std::cout (0405078h)]
00401063  push        ecx
00401064  call        std::operator<<<std::char_traits<char> > (0401530h)
00401069  add         esp,8
0040106C  lea         edx,[input1]
0040106F  push        edx
00401070  mov         eax,dword ptr [__imp_std::cin (0405074h)]
00401075  push        eax
00401076  call        std::operator>><char,std::char_traits<char> > (0401500h)
0040107B  add         esp,8
```

```
    // Check if the input is equal to the password
    int result;
    result = strcmp(password, input1);
0040107E  lea         ecx,[input1]
00401081  push        ecx
00401082  lea         edx,[password]
00401085  push        edx
00401086  call        _strcmp (0404291h)
0040108B  add         esp,8
0040108E  mov         dword ptr [result],eax

    if (result == 0)
00401091  cmp         dword ptr [result],0
00401095  jne         CheckPass+99h (04010B9h)
        cout << "Valid access" << endl;
00401097  push        offset std::endl<char,std::char_traits<char> > (0401AC0h)
0040109C  push        4053F4h
004010A1  mov         eax,dword ptr [__imp_std::cout (0405078h)]
004010A6  push        eax
004010A7  call        std::operator<<<std::char_traits<char> > (0401530h)
004010AC  add         esp,8
004010AF  mov         ecx,eax
004010B1  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (0405058h)]
004010B7  jmp         CheckPass+0C2h (04010E2h)
    else
    {
        cout << "Stop there!" << endl;
004010B9  push        offset std::endl<char,std::char_traits<char> > (0401AC0h)
004010BE  push        405404h
004010C3  mov         ecx,dword ptr [__imp_std::cout (0405078h)]
004010C9  push        ecx
004010CA  call        std::operator<<<std::char_traits<char> > (0401530h)
004010CF  add         esp,8
004010D2  mov         ecx,eax
004010D4  call        dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (0405058h)]
        exit(0);
004010DA  push        0
004010DC  call        dword ptr [__imp__exit (0405130h)]
    }
```

As we see in the picture, the instruction address for the epilogue is 004010DA.

Machine code of the first assembly instruction inserted in the inline assembly of CheckInlineAssembly().

```
int solution = 0;

__asm {
    xor ecx, ecx
    mov eax, integerSuperior;    eax = integerSuperior
    mov ebx, integerInferior;    bx = integerInferior
    cmp eax, ebx;                we compae if they are equal
    jne consequent;              different
    jmp end;                     if they are equal finish
    consequent :
    inc ecx;                     we increment the value by one in case they are different
        mov solution, ecx;       store that value on solution
    end :
}

if (solution > 0)
{
    cout << "Something went wrong";
    exit(0);
}
```

First, we perform a *xor* operation to free up the register, just in case it already was storing a value. Then, we move both parts of the integer to registers and we compare them. In the case that they are equal, we jump to the end. If not, we jump to consequent, where we increment the previously freed register and move its value to the variable “solution”.

Finally, outside the inline assembly we compare if solution has been increased, in which case we print the statement “Something went wrong” and we exit the program.

Valid and invalid inputs

CheckPass()

```
Consola de depuración de Microsoft Visual Studio
Input the password: 43WscadOB
Valid access
Give me another string: helloWorld
Correct

U:\USB\PRIMERO\FCR\Teamwork\Teamwork\Debug\Teamwork.exe (pr
Presione cualquier tecla para cerrar esta ventana. . .
```

```
Consola de depuración de Microsoft Visual Studio
Input the password: hi
Stop there!
```

```
Consola de depuración de Microsoft Visual Studio
Input the password: 43WscadOB
Valid access
Give me another string: hey
Access denied
```

CheckAccessBits()

```
Consola de depuración de Microsoft Visual Studio
Input1 the first integer password: 32768
Input2 the second integer password: 8
Correct
```

Consola de depuración de Microsoft Visual Studio

```
Input1 the first integer password: 0
Input2 the second integer password: 3840
You're not allowed here
```

Consola de depuración de Microsoft Visual Studio

```
Input1 the first integer password: 0
Input2 the second integer password: 1
Intruder detected
```

Consola de depuración de Microsoft Visual Studio

```
Input1 the first integer password: 67584
Input2 the second integer password: 2056
Bad luck
```

AsmAcess()

Consola de depuración de Microsoft Visual Studio

```
Input1 a 32-bit integer: 2048
Input2 a 32-bit integer: 0
Input3 a 32-bit integer: 988921
Correct
```

Consola de depuración de Microsoft Visual Studio

```
Input1 a 32-bit integer: 3840
Input2 a 32-bit integer: 0
Input3 a 32-bit integer: 988921
Bad luck
```

Consola de depuración de Microsoft Visual Studio

```
Input1 a 32-bit integer: 3840
Input2 a 32-bit integer: 0
Input3 a 32-bit integer: 0
Bad luck
```

CheckInlineAssembly()

```
Consola de depuración de Microsoft Visual Studio
Give me a 32-bit integer: 1

Something went wrong
```

```
Consola de depuración de Microsoft Visual Studio
Give me a 32-bit integer: 0

Correct
```

Work division

We decide to work together via Teams meeting, the code was implemented in various sessions. On the first session, Jorge and Mikel started implementing the CheckPass() method while Miguel was working on the CheckAccessBits(), later on we would revise each other´s code and see if something needed to be changed, in case it wasn´t working or there were better ways to implement it. We would constantly exchange different versions of the method and finally implement a common one that resembled both. Then, we would all finish the second method.

Since we didn´t know much about assembly yet, we decided to continue another day since we were also starting to study the instructions in the theory classes. Once we understood the concepts and watch all the videos provided to us by Joaquín, we started to work on the two remaining methods AsmAcess() and CheckInLineAssemblyAccess(). At first, we had a couple of doubts on how to start these methods, but we would later figure it all out. We also had to read the Intel Architecture, since some conditions had a different name from the ones seen in class or the way that we are supposed to name registers.

Finally, we reunited another afternoon night to conclude with the first phase by checking if everything worked as expected, provide the results after running the program by taking screenshots and last answer all the questions.