Web Security and Malware Analysis

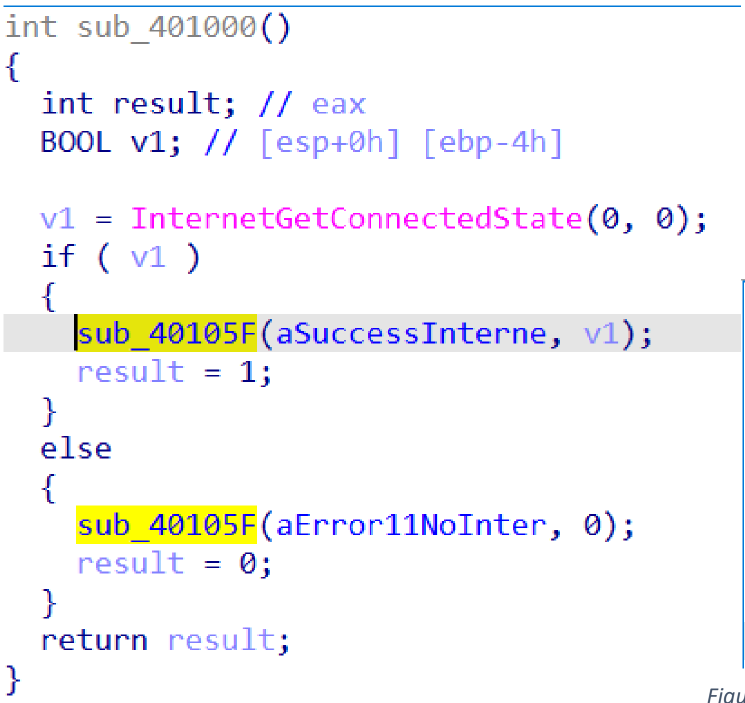
Answers for Assignment 7

SILVIA LUCIA SANNA – 70/90/00053

Task 1 – Static Malware Analysis

To make a static malware analysis we have 2 different approaches: forward and backward analysis. I decided to perform both, starting from forward.

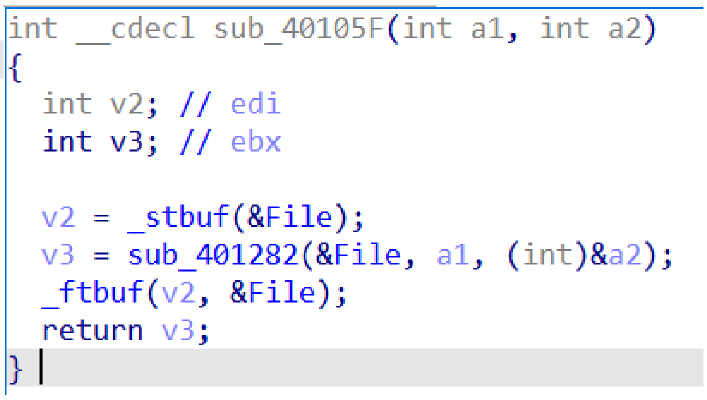
Here I select the first function in IDA which is main 0x00401040, I search it in Ghidra and found that it calles a function 401000. By analysing it I found that it makes an internet connection checking the state, so if the connection is established it will like print a message of success, otherwise it will print error. I thought that it prints a message because analysing the decompiled function 40105f calls 3 functions one after the other: _stbuf, 401282, _ftbuf. The function 401282 is very long but looking briefly it makes something with strings. Searching on Google and analysing the input variables, I can assume that function 40105f is a kind of printf. In fact, it receives in input a string for correct or incorrect connection which makes think that prints this message.



```
int sub_401000()
{
  int result; // eax
  BOOL v1; // [esp+0h] [ebp-4h]

  v1 = InternetGetConnectedState(0, 0);
  if ( v1 )
  {
    sub_40105F(aSuccessInterne, v1);
    result = 1;
  }
  else
  {
    sub_40105F(aError11NoInter, 0);
    result = 0;
  }
  return result;
}
```

Figure 1: decompiled code for 401000



```
int __cdecl sub_40105F(int a1, int a2)
{
  int v2; // edi
  int v3; // ebx

  v2 = _stbuf(&File);
  v3 = sub_401282(&File, a1, (int)&a2);
  _ftbuf(v2, &File);
  return v3;
}
```

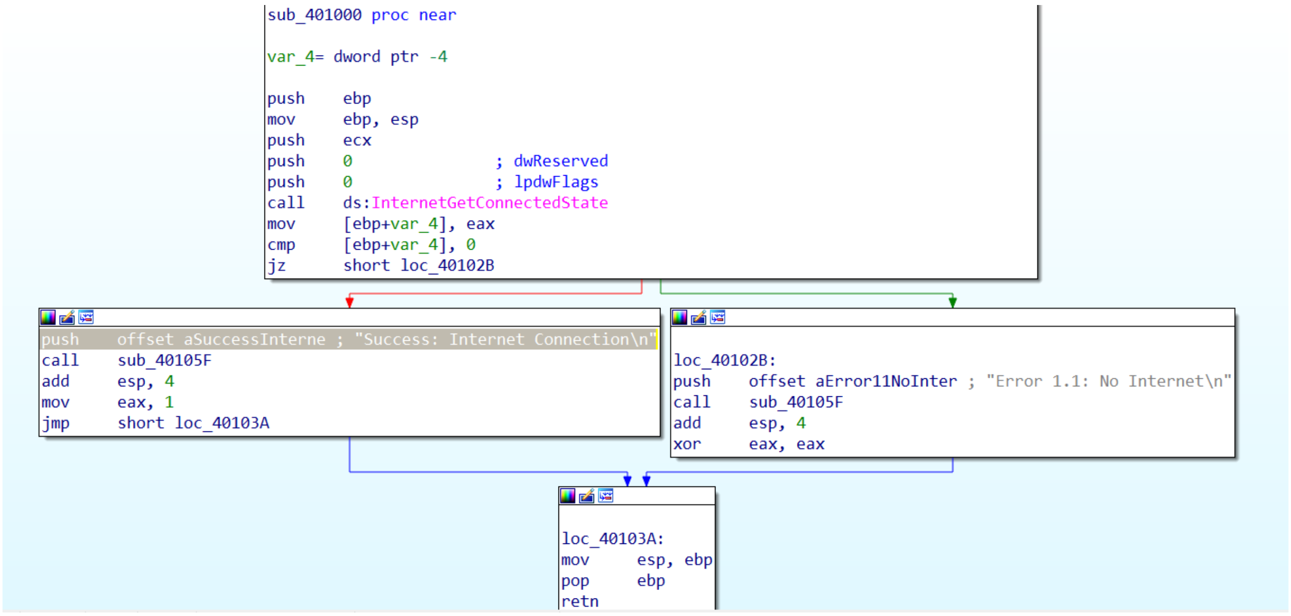
Figure 2: decompiled code for 40105F, renamed as printf

So, making a forward analysis we can see in the main function only a call to 401000 function which in turn calls two functions:

- <u>InternetGetConnectedState</u> an imported function from some library, maybe wininet, to check if there is an internet connection and depending on that returns 0 or 1.
- <u>40105F</u> (renamed printf) to print the message related to the connection status which is also set to the variable result.

In the main, the value in result variable will be compared to 0 (checking if equal or not).

I can confirm this theory with the backward analysis: checking the strings only 2 are interesting for this case, one for internet connection error and the other one for the success. Looking the cross references, we see that both are called in function 401000 and are contained in two different branches depending on the comparison between 0 and a variable which is the output of InternetGetConnectedState.



```
sub_401000 proc near

var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
push    0               ; dwReserved
push    0               ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40105F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

```
loc_40102B:
push    offset aError11NoInter ; "Error 1.1: No Internet\n"
call    sub_40105F
add     esp, 4
xor     eax, eax
```

```
loc_40103A:
mov     esp, ebp
pop     ebp
retn
```

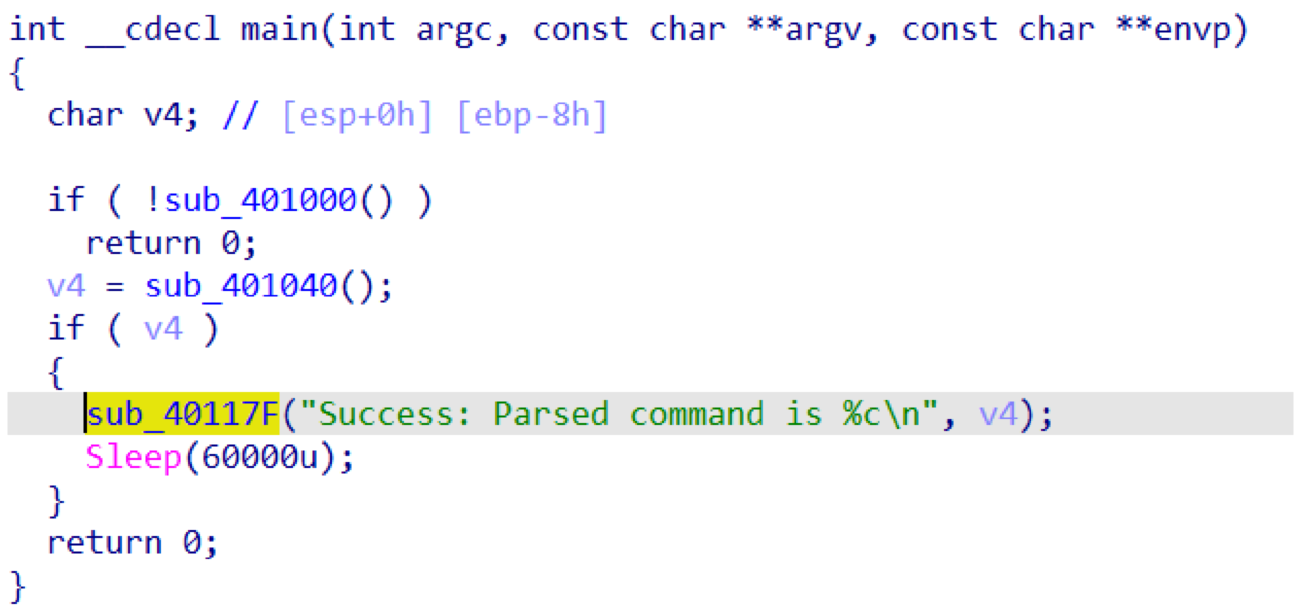
In conclusion, from this analysis I can say that this malware only checks the connection status.

Task 2 – More Static Analysis

Again, in this task I have to statically analyse a malware and again I choose to make a forward and backward analysis.

Here the main starts at 401130 and calles 4 functions:

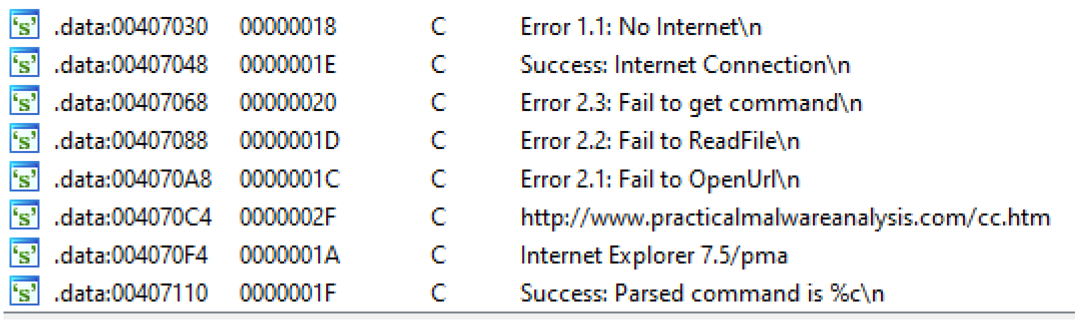
- 401000 which checks the Internet connection state using an imported function, prints the message for correct or incorrect connection (function 40117f). This last printf is different from the one found in the previous task, it is much longer but is composed by 3 functions like in the previous task and two of them are called _stbuf and _ftbuf; another thing that makes me think about printf is that all functions manipulate same strings/parameters. What I did not expected to find, as in the previous exercise, are some functions here calling to free the heap.
- 401040 which opens Internet (browser Internet Explorer 7.5), a URL (practicalmalwareanalysis.com) and reads a file checking if it contains "<!--", the first 4 characters read from Buffer (I can deduce that is an HTML file because this characters in HTML stands for comments). If the file contains those characters, will set the return variable result to the value contained in the 5th position of the array Buffer, whose value can be seen only with a dynamic analysis. In case of any unsatisfied "if" it will close the connection.
- 40117f which is the function contained in the 401000 function and that in my opinion makes a prinft. This theory is confirmed looking the decompiled code in the main: this function receives in input a string "Success" and a variable, like I will do in a printf or even puts.
- Sleep where the program will sleep for 1 minute.


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [esp+0h] [ebp-8h]

  if ( !sub_401000() )
    return 0;
  v4 = sub_401040();
  if ( v4 )
  {
    sub_40117F("Success: Parsed command is %c\n", v4);
    Sleep(60000u);
  }
  return 0;
}
```

Figure 3: decompiled code with IDA for main function

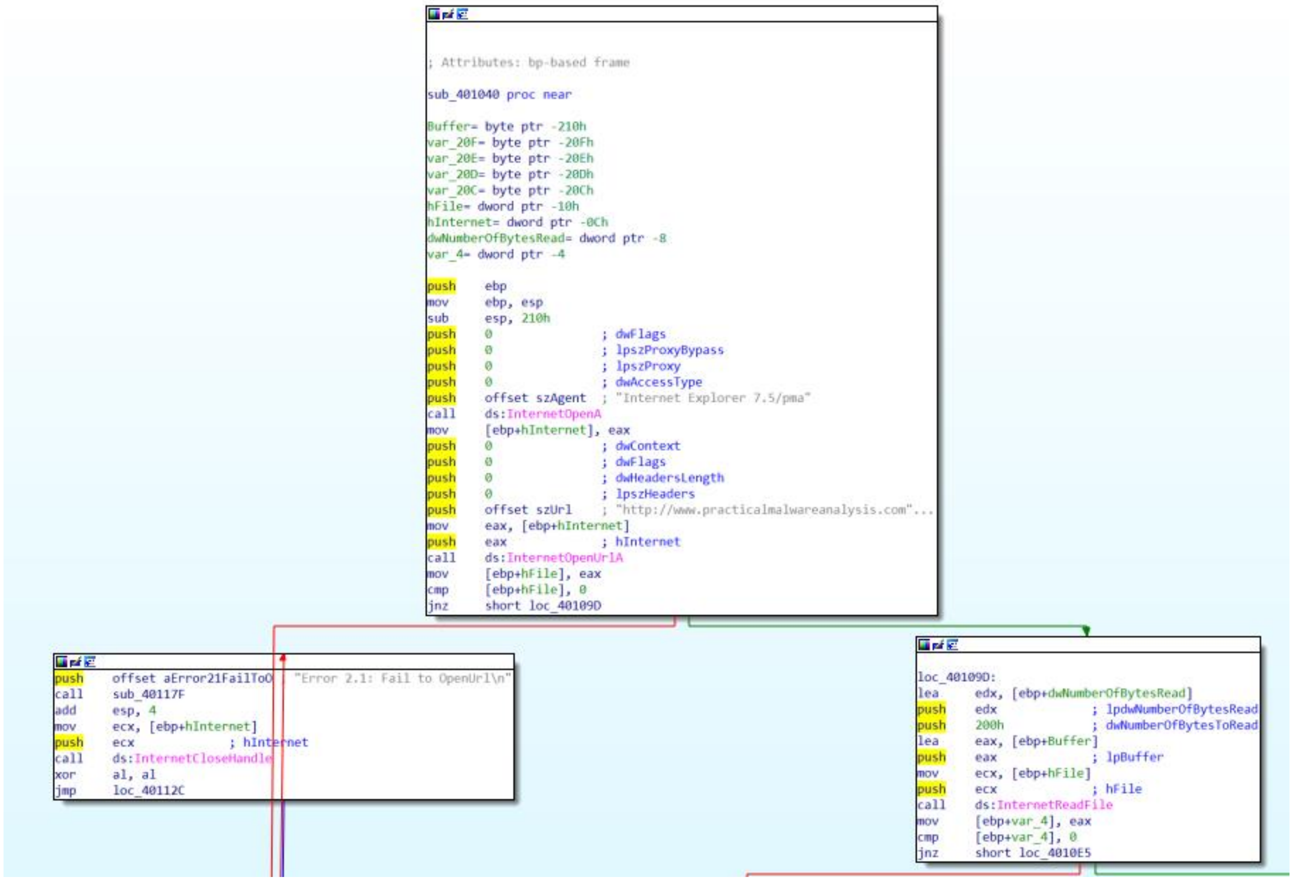
I can confirm all this theories making a backward analysis: in the string session on IDA I found 8 interesting strings, so I looked all their cross references and I reconstructed the exact flow I saw in the forward analysis.



.data:00407030	00000018	C	Error 1.1: No Internet\n
.data:00407048	0000001E	C	Success: Internet Connection\n
.data:00407068	00000020	C	Error 2.3: Fail to get command\n
.data:00407088	0000001D	C	Error 2.2: Fail to ReadFile\n
.data:004070A8	0000001C	C	Error 2.1: Fail to OpenUrl\n
.data:004070C4	0000002F	C	http://www.practicalmalwareanalysis.com/cc.htm
.data:004070F4	0000001A	C	Internet Explorer 7.5/pma
.data:00407110	0000001F	C	Success: Parsed command is %c\n

In conclusion, this malware checks the Internet connection and if success, makes a connection to a specific url practicalmalwareanalysis.com with the browser Internet Explorer 7.5. In this webpage will read the html file and get the 5th character, prints it and will wait 1 minute. As the sample in the previous task, here there is an internet connection check but after it there are some more actions.

To be sure that I am right with what happens in function 401040, I can analyse the assembly which can be divided in 3 parts.



```
; Attributes: bp-based frame

sub_401040 proc near

Buffer= byte ptr -210h
var_20F= byte ptr -20Fh
var_20E= byte ptr -20Eh
var_20D= byte ptr -20Dh
var_20C= byte ptr -20Ch
hFile= dword ptr -10h
hInternet= dword ptr -0Ch
dwNumberOfBytesRead= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 210h
push    0               ; dwFlags
push    0               ; lpszProxyBypass
push    0               ; lpszProxy
push    0               ; dwAccessType
push    offset szAgent  ; "Internet Explorer 7.5/pma"
call    ds:InternetOpenA
mov     [ebp+hInternet], eax
push    0               ; dwContext
push    0               ; dwFlags
push    0               ; dwHeadersLength
push    0               ; lpszHeaders
push    offset szUrl    ; "http://www.practicalmalwareanalysis.com"...
mov     eax, [ebp+hInternet]
push    eax             ; hInternet
call    ds:InternetOpenUrlA
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0
jnz     short loc_40109D
```

```
push    offset aError21FailToO ; "Error 2.1: Fail to OpenUrl\n"
call    sub_40117F
add     esp, 4
mov     ecx, [ebp+hInternet]
push    ecx             ; hInternet
call    ds:InternetCloseHandle
xor     al, al
jmp     loc_40112C
```

```
loc_40109D:
lea     edx, [ebp+dwNumberOfBytesRead]
push    edx             ; lpdwNumberOfBytesRead
push    200h            ; dwNumberOfBytesToRead
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
mov     ecx, [ebp+hFile]
push    ecx             ; hFile
call    ds:InternetReadFile
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz     short loc_4010E5
```

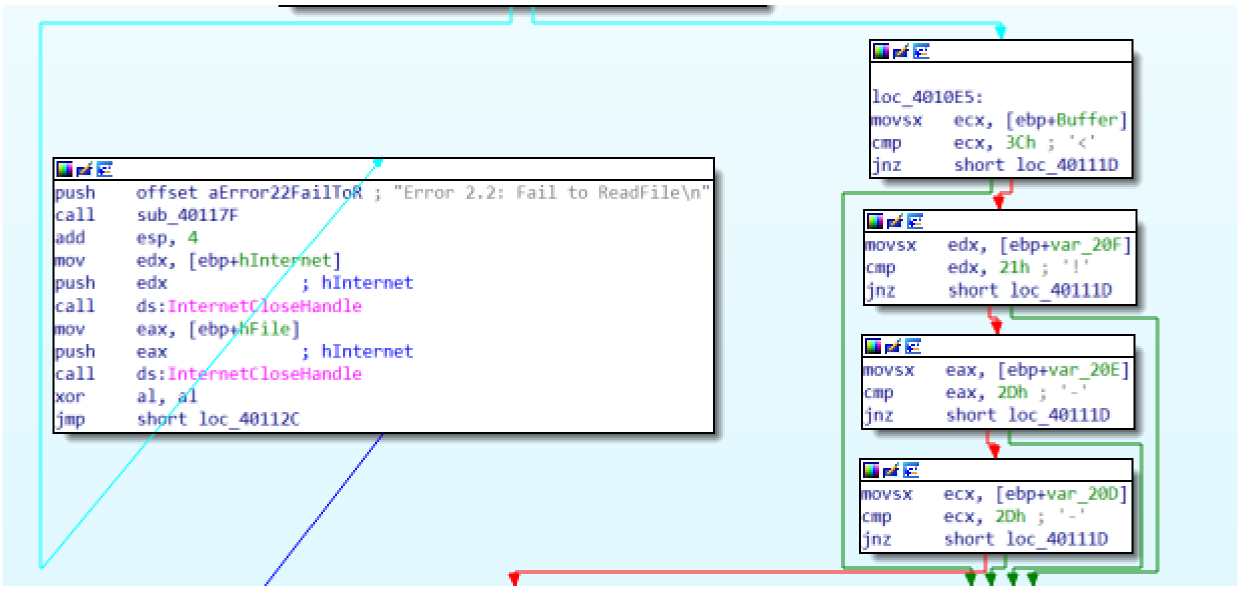
Figure 4: Part 1 of function 401040 containing the main and the two blocks for true (right) and false (left) branch

In the image above we see the first part of the function divided in 2 blocks, the first one at the top and the two blocks regarding the false and true branch of the if condition.

In the first block there is the initialization of some variables (first 9 lines), where the first one is the address of the first element while the next ones are the next elements, in fact the first element is the highest value on the stack. After this there is the stack preparation (first 3 lines of instructions), puts 5 parameters on stack and calls to the imported function "Internet Open" whose result is moved from variable eax to ebp+hInternet (which is a store). After that we have the load of other 5 parameters on stack and 1 more which is the one stored in ebp+hInternet that will be put again in register eax. Now there is the call to function InternetOpenUrl imported from some library and its result stored in ebp+hFile.

After this there is the comparison between this variable and the value 0, if they are not equal (so if the variable is not zero), the jump will be taken and so I have to analyse the block in the green arrow. Otherwise the block on the red arrow will be taken (there is like a double arrow because it was very down with respect to the green one, so I brought it up).

- In the green branch (right), in the first line we have the store of the effective address of ebp+dwNumberOfBytesRead in edx which is then stored on top in the stack together with dwNumberOfBytesToRead. This is done for lpBuffer and then a move and store in stack for hFile parameters. After that InternetReadFile function is called whose result from eax is stored on ebp+var4 and compared to zero and if they are not equal will jump to loc 4010E5.
- In the red branch (left), the first instruction loads in the stack the string “Error Fail to open url”. Then the printf function is called to print this string and the stack pointer is incremented by 4 (if I am not wrong this addition is made because a new value is added in the stack). After this, the parameter hInternet is loaded in ecx register and passed to function InternetCloseHandle and then a xor in the al variable is made (to know its value I think that I should make a debug but making a xor with the same variable, means to set it to 0) and an immediately jump to loc 40112C.



```
push    offset aError22FailToR ; "Error 2.2: Fail to ReadFile\n"
call    sub_40117F
add     esp, 4
mov     edx, [ebp+hInternet]
push    edx             ; hInternet
call    ds:InternetCloseHandle
mov     eax, [ebp+hFile]
push    eax             ; hInternet
call    ds:InternetCloseHandle
xor     al, al
jmp     short loc_40112C
```

```
loc_4010E5:
movsx   ecx, [ebp+Buffer]
cmp     ecx, 3Ch ; '<'
jnz     short loc_40111D
```

```
movsx   edx, [ebp+var_20F]
cmp     edx, 21h ; '!'
jnz     short loc_40111D
```

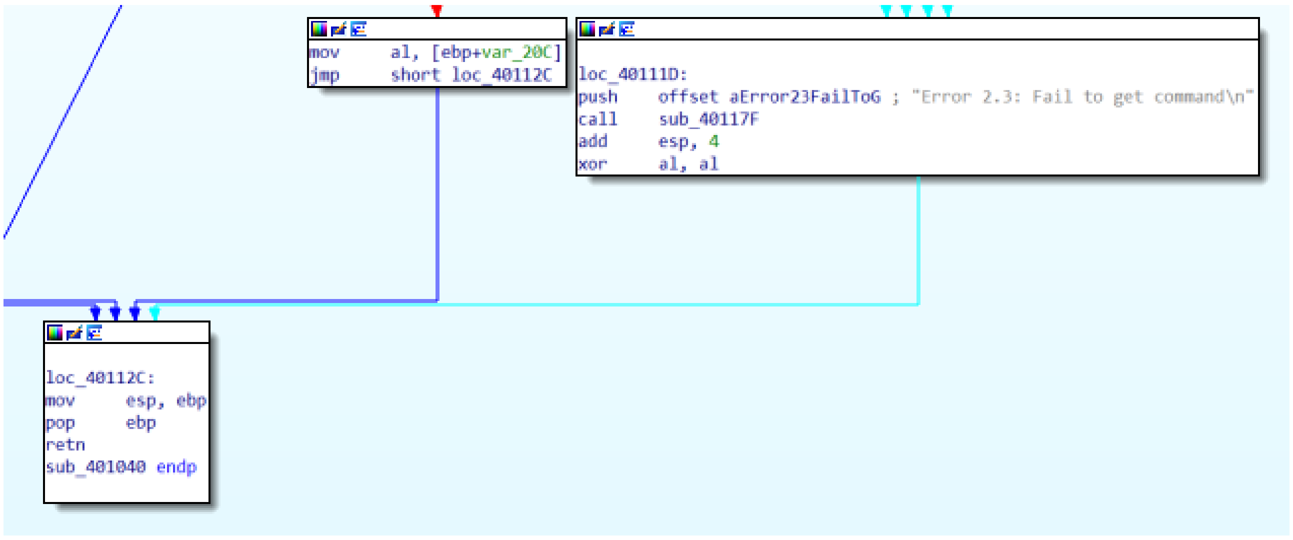
```
movsx   eax, [ebp+var_20E]
cmp     eax, 2Dh ; '-'
jnz     short loc_40111D
```

```
movsx   ecx, [ebp+var_20D]
cmp     ecx, 2Dh ; '-'
jnz     short loc_40111D
```

Figure 5: Part 2 of function 401040 composed by true (right) and false (left) branches of loc_40109D

In the second block of function 401040 we have the right and false branch of location 40109D analysed here above.

- In the green branch (right) there are 4 equal blocks repeated one after the other: in all of them there is the load of a certain element of the array Buffer in a specific register (ecx, edx, eax, ecx) and a comparison with the hexadecimal of a character (< ! - -). If this comparison is correct it will pass to the next block (red arrow) and in the last block will go to the left block in the image above. If the comparison is not right, it will jump to <u>loc 40111D</u>. The structure I have just described is a classical representation of a nested if and the variables to be compared loaded in specific registers are elements of array Buffer (ebp+Buffer means the first element of the array, while the other are stored as variables as described in the first lines).
- In the red branch (left), first of all there is the load of string "Error fail to read file" which will be passed to printf (function 40117F) and the stack pointer incremented by 4. After this parameter hInternet is loaded in edx register and InternetCloseHandle function is called. Then hInternet is loaded in eax register and InternetCloseHandle is called again. Here again there is the xor with al and jump immediately to <u>loc 40112C</u>.



```
mov     al, [ebp+var_20C]
jmp     short loc_40112C
```

```
loc_40111D:
push    offset aError23FailToG ; "Error 2.3: Fail to get command\n"
call    sub_40117F
add     esp, 4
xor     al, al
```

```
loc_40112C:
mov     esp, ebp
pop     ebp
retn
sub_401040 endp
```

Figure 6: Part 3 of function 401040, composed by true (first line on the right) and false branches (first line on the left) of loc_4010E5 and last block at the bottom to the end of function 401040.

- In the first block at the top left, there are the instructions executed after all comparisons to characters

"<!--" are satisfied. Here the value of ebp+var_20C is loaded in al and jump to loc 40112C.

- Otherwise if the comparison to even only one of the characters is not satisfied instructions in loc 40111D will be executed. The string "Error Fail to get command" is loaded and passed to the printf 40117F and the stack pointer incremented by 4. Again, a xor with al is made and to know its value some debug should be made but as is a xor with itself I can assume that is set to 0 (al=0).

After all we reach location 40112C which is the end of the program, in fact there is the realignment between stack and base pointer and the removal of stack pointer.

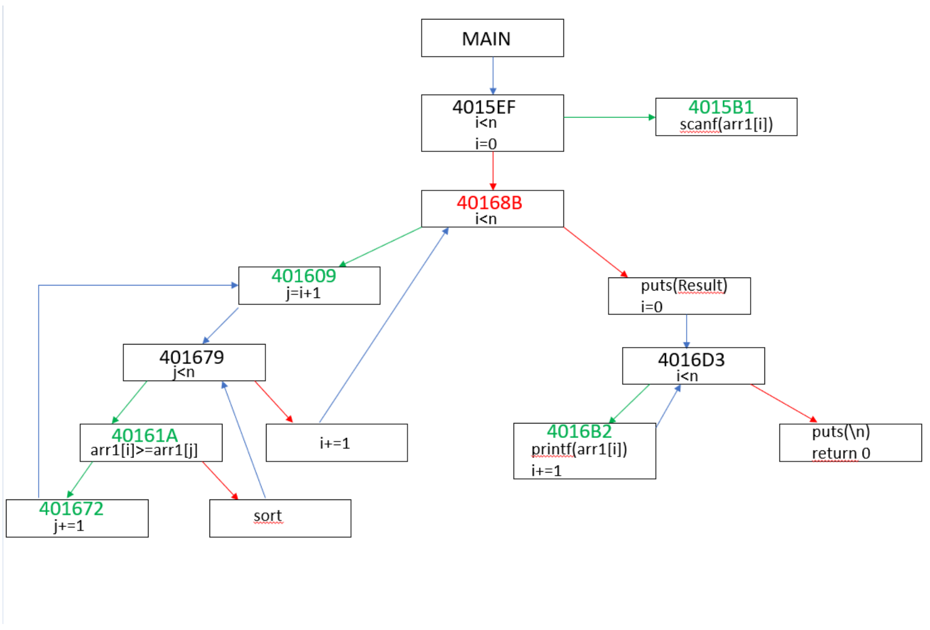
Task 3 – Assembly X64 Analysis

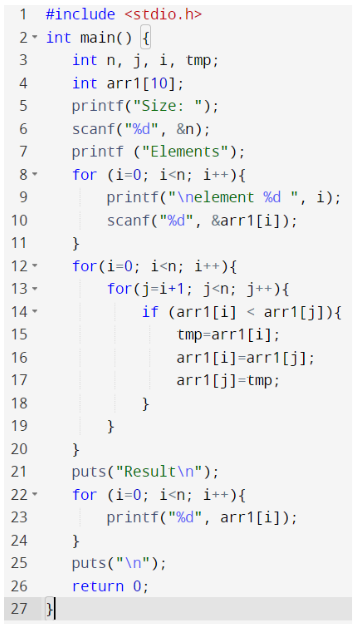
In this task I have to analyse the assembly x64 not line by line but only in general how it works, so I decided to analyse per blocks (between many dashes symbols). According to this criterion, there are 5 blocks and I will analyse them following the program flow:

- The first block is the start of the function main. In the first lines there is the initialization of 5 parameters (n, arr1, tmp, j, i), then the stack organization. After that a message is prompt to the user asking to insert a size, then the user will insert a number, set the value of variable i to 0 and move to loc_4015EF.
- In loc 4015EF I see the structure of a for loop because there is the comparison less than between variable i and n. So, if i<n will jump to loc_4015B1, otherwise will set i=0 and jump to 40168B.
- In loc 4015B1 the user will insert all n elements, I deduce it because there is a printf asking for element in i-th position and a relative scanf with a pointer to arr1 in the relative i-th position (the element arr1[i] is located at arr1+i*4 because each slot is made by 4 bytes).
- In loc 40168B variable i will be compared to variable n and if i<n will jump to 401609, otherwise calls puts with as

argument the string Result and after setting i=0 will jump to 4016D3.

- In loc_401609 variable i is incremented by 1 and variable j will assume its value, so j=i+1 and jump to 401679.
- In loc_401679 if j<n jump to 40161A, otherwise add i+=1
- In loc_40161A there is the comparison between consecutive elements of arr1. In fact, we have two indexes i and j, this last one is j=i+1. If the element i-th is less than the j element there is a shift between these elements, like to order the array in decreasing way. If the element i-th is greater or equal than element j-th it will jump to 401672.
- In loc_401672 j is incremented by 1 to continue in that cycle.
- In loc_4016D3 if i<n jump to 4016B2, otherwise call puts with “\n”, restore the stack and end.
- In loc_4016B2 will be print arr1[i] and I incremented by 1.





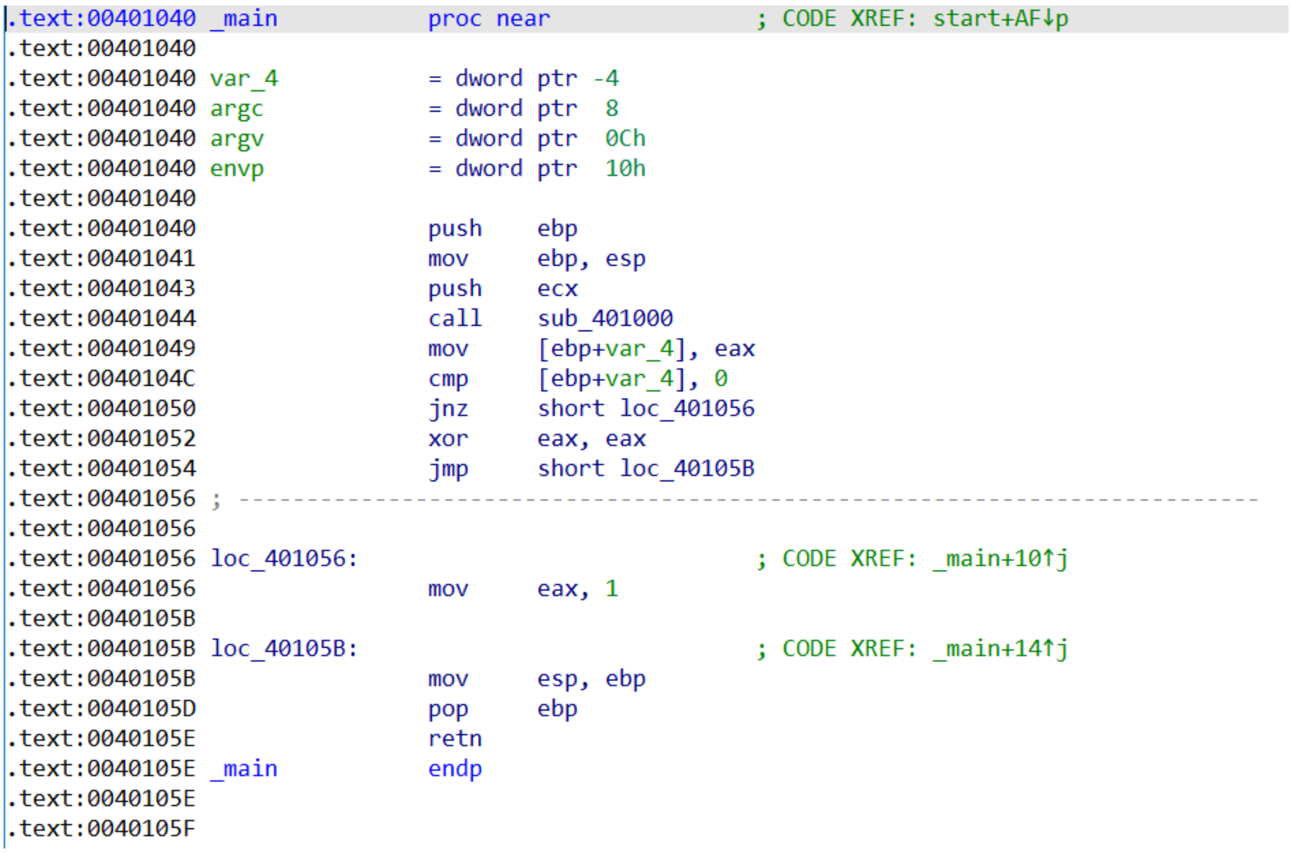
```
#include <stdio.h>
int main() {
    int n, j, i, tmp;
    int arr1[10];
    printf("Size: ");
    scanf("%d", &n);
    printf ("Elements");
    for (i=0; i<n; i++){
        printf("\nelement %d ", i);
        scanf("%d", &arr1[i]);
    }
    for(i=0; i<n; i++){
        for(j=i+1; j<n; j++){
            if (arr1[i] < arr1[j]){
                tmp=arr1[i];
                arr1[i]=arr1[j];
                arr1[j]=tmp;
            }
        }
    }
    puts("Result\n");
    for (i=0; i<n; i++){
        printf("%d", arr1[i]);
    }
    puts("\n");
    return 0;
}
```

Figure 7: On the left there is the representation in code blocks of the assembly code, as IDA does. Green arrows and locations mean that the branch is true. Red arrows and locations mean the branch is false. Blue arrows mean that immediately goes to it. In each block there is the main of the location if present and in black what happens in that block. On the right there is the C code for this assembly.

Looking at these schema is easy to recognise some advanced constructs: first of all arrays (arr1), then nested for (for in j inside for in i) and a lot of if statements (check the index and check the elements).

BONUS TASK – A strange function

In the following part of assembly x86 code, we can see at 401040 the input parameters to main function, then in the following 3 lines (401040 push ebp – 401043) stack preparation; the call to sub_401000 which I think will return eax to be put in var_4 and compared with 0 (401049 – 40104C) and if not 0 will jump to loc_401056 where eax=1; otherwise (if var_4=0) will make a xor operation in eax (make eax = eax^eax means to set eax to 0) and jump immediately to loc_40105B where there is the restore of stack and end of the procedure.



```
.text:00401040 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401040
.text:00401040 var_4           = dword ptr -4
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    ecx
.text:00401044                 call    sub_401000
.text:00401049                 mov     [ebp+var_4], eax
.text:0040104C                 cmp     [ebp+var_4], 0
.text:00401050                 jnz     short loc_401056
.text:00401052                 xor     eax, eax
.text:00401054                 jmp     short loc_40105B
.text:00401056 ; ---------------------------------------------------------------------------
.text:00401056
.text:00401056 loc_401056:                             ; CODE XREF: _main+10↑j
.text:00401056                 mov     eax, 1
.text:0040105B
.text:0040105B loc_40105B:                             ; CODE XREF: _main+14↑j
.text:0040105B                 mov     esp, ebp
.text:0040105D                 pop     ebp
.text:0040105E                 retn
.text:0040105E _main           endp
.text:0040105E
.text:0040105F
```

Figure 8: Assembly code from 401040 up to 40105F

So, this routine will call a specific function 401000 which I do not know what specifically does but for sure returns a variable in register eax. If the returned value is 0 will end, otherwise returns 1, so this function checks the Boolean value of the returned value of 401000. I do not know how code of next subroutines will be only analysing this part so I am not so able to say why it would be so complex.