



Reverse Engineering

Lab 06

Timo Lehosvuori, M3426

Report

Reverse Engineering, Marko Silokunnas

7.3.2021

ICT

Sisällys

1	Lab 06.....	3
2.	Time spent.....	7

1 Lab 06

Unlike the previous labs there were a lot going on in the main function, but the main point is the “_memcpy” where it copies “109” bytes from the location of “byte_8048810” and at the end where it sets “var_B0” to “0”. It also sets “30” bytes to “0” in location of “var_AA” but this was not essential to solve the lab:

```

mov     [ebp+var_B8_arg4], eax
lea     eax, byte_8048810
mov     [ebp+var_BC_ptr_unk804], eax
mov     eax, 109
mov     [ebp+var_C0_109], eax
lea     eax, [ebp+var_85]
mov     [ebp+var_10], 0
mov     [ebp+var_14_arg0], ecx
mov     ecx, [ebp+var_B8_arg4]
mov     [ebp+var_18_arg4], ecx
mov     [esp+118h+var_118_ptr_var85], eax
mov     eax, [ebp+var_BC_ptr_unk804]
mov     [esp+118h+var_114_ptr_unk804], eax
mov     [esp+118h+var_110], 109
mov     [ebp+var_C4_ptr_varAA], ebx
mov     [ebp+var_C8_ptr_apassw], edx
mov     [ebp+var_CC esiXOR], esi
mov     [ebp+var_D0_30], edi
call    _memcpy          ; d: var85, s: unk804, 109
mov     [ebp+var_8C], 109
mov     eax, [ebp+var_C4_ptr_varAA]
mov     [esp+118h+var_118_ptr_var85], eax
mov     [esp+118h+var_114_ptr_unk804], 0
mov     [esp+118h+var_110], 30
call    _memset          ; ptr: varAA, value: 0, bytes: 30
mov     eax, [ebp+var_C8_ptr_apassw]
mov     [esp+118h+var_118_ptr_var85], eax
call    _printf          ; prints content of apassword
lea     ecx, aS          ; "%s"
lea     edx, [ebp+var_AA]
mov     [esp+118h+var_118_ptr_var85], ecx ; ptr to aS
mov     [esp+118h+var_114_ptr_unk804], edx ; ptr to varAA
mov     [ebp+var_D4], eax ; ptr to apassword
call    _isoc99_scanf
mov     [ebp+var_B0], 0
mov     [ebp+var_D8], eax

```

Figure 1: Main function.

Afterwards it jumps to “checker” that checks if the value of “eax” is 109 and if not, it goes to a loop:

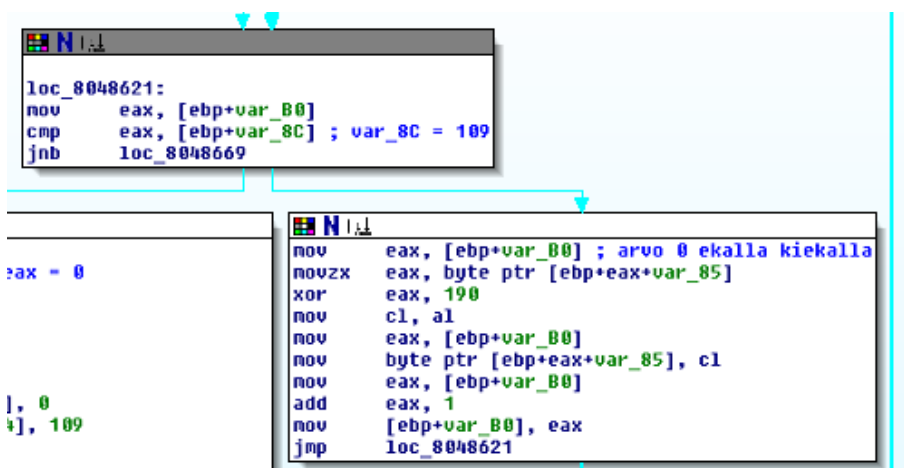


Figure 2: check_serial function.

The loop takes the byte indicated by the value of “eax” (in the first run its 0) from the location of “var_85” and it XORs this with 190. The output of this is then put to 8-bit data register “al” which is moved to “cl” and later in the code it is moved back to the location of “var_85” and to the same byte index indicated by the value of “eax”. Basically, this loop XORs the values in “var_85” and puts the XORed values back to “var_85”. Value of “eax” is incremented by one and the loop starts over until “eax” is “109”. When the value of “eax” is “109” the code moves to a different section where a lot of interesting things happen:

```

loc_8048669:                ; xor eax, eax = 0
xor     eax, eax
mov     ecx, 109
mov     edx, 7
mov     esi, 34
mov     edi, 4294967295
mov     [esp+118h+var_118_ptr_var85], 0
mov     [esp+118h+var_114_ptr_unk804], 109
mov     [esp+118h+var_110], 7
mov     [esp+118h+var_10C], 34
mov     [esp+118h+var_108], 4294967295
mov     [esp+118h+var_104], 0
mov     [ebp+var_0C], eax ; 0
mov     [ebp+var_E0], ecx ; 109
mov     [ebp+var_E4], edx ; 7
mov     [ebp+var_E8], esi ; 34
mov     [ebp+var_EC], edi ; 4294967295
call    _mmap                ; eax osottaa mmap?
lea     ecx, [ebp+var_AA]
mov     edx, 109
lea     esi, [ebp+var_85]
mov     [ebp+var_B4], eax
mov     eax, [ebp+var_B4]
mov     [esp+118h+var_118_ptr_var85], eax
mov     [esp+118h+var_114_ptr_unk804], esi
mov     [esp+118h+var_110], 109
mov     [ebp+var_F0], edx
mov     [ebp+var_F4], ecx
call    _memcpy              ; d: mmap, s: var85, bytes: 109
mov     eax, [ebp+var_B4] ; eax = mmap osote
mov     ecx, esp            ; ecx ptr to stack
mov     edx, [ebp+var_F4] ; edx ptr to var_AA
mov     [ecx], edx          ; var_AA osotteen sisältö stäkkiin
mov     [ebp+var_F8], eax ; memcpy tulos
call    _strlen
mov     [esp+118h+var_118_ptr_var85], eax
mov     eax, [ebp+var_F4]
mov     [esp+118h+var_114_ptr_unk804], eax
mov     ecx, [ebp+var_F8]
call    ecx
cmp     eax, 0
jnz     loc_8048760

```

Figure 3: check_serial function loops.

The functions that I needed to solve to solve the lab are “_mmap”, “_memcpy” and I needed to figure out what is inside “ecx” at the end where it is called. The “_mmap” function maps files or devices into memory and in this case it takes 6 arguments: addr (0) tells the starting address of the mapping, length (109) tells the length of the mapping, prot (7) describes memory protection, flags (34) this tells whether updates to the mapping are visible to other processes and are they carried to underlying files, file descriptor (4294967295) and offset (0) tells where the “length” starts

(“mapping starts at 0 with length of 109”). This new mapped area is used at the function “_memcpy” where it copies “109” bytes from “var_85” (XORed content from earlier) to the mapped area pointed by “mmap” (mmap returns a pointer to the mapped area). Now that I started to have an understanding what’s going on, I still did not know what the content inside “mmap” is used for. I knew that the code calls “ecx” which holds the result of “_memcpy” so I figured that it had to hold a function inside. I went and copied the values inside “byte_8048810” and XORed them using cy-berchef:

Figure 4: HEX values.

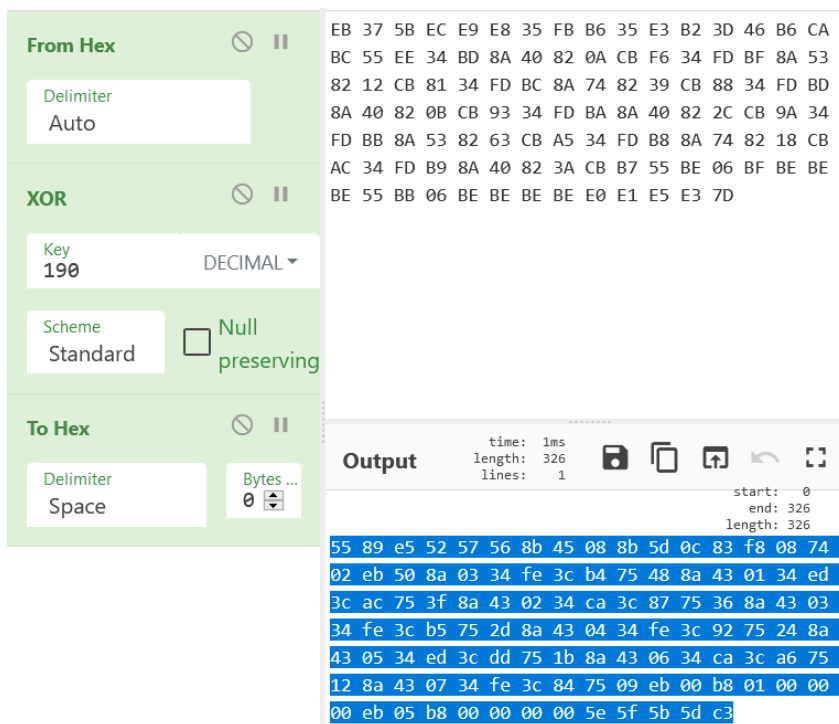


Figure 5: HEX after XOR.

Now that I had the content of the “mmap” memory I had to disassemble this somehow, so I googled “online disassembler” and use the first one I found (<https://onlinedisassembler.com/static/home/index.html>):

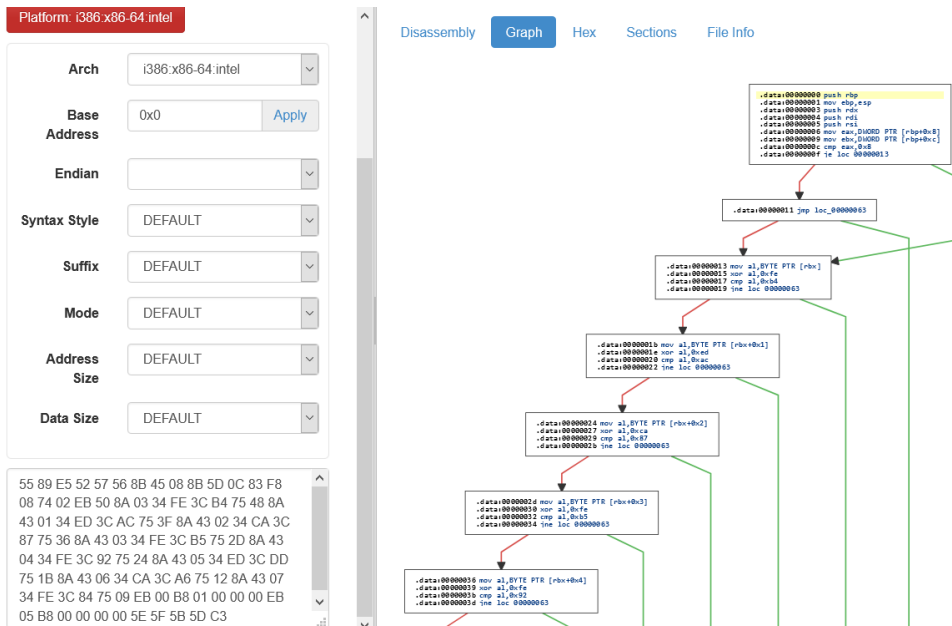


Figure 6: Online disassembler.

I changed the “Arch” to i386:x86-64:intel and started to look at the result. The code starts with some “push”, “mov” and “cmp” instructions until it jumps to a loop and the loop interested me the most so I shifted my focus there. I realised that it did some XORing and just like in the previous labs I “reverse” XORed the inputs in each loop and got the result “JAMKIOlz”:

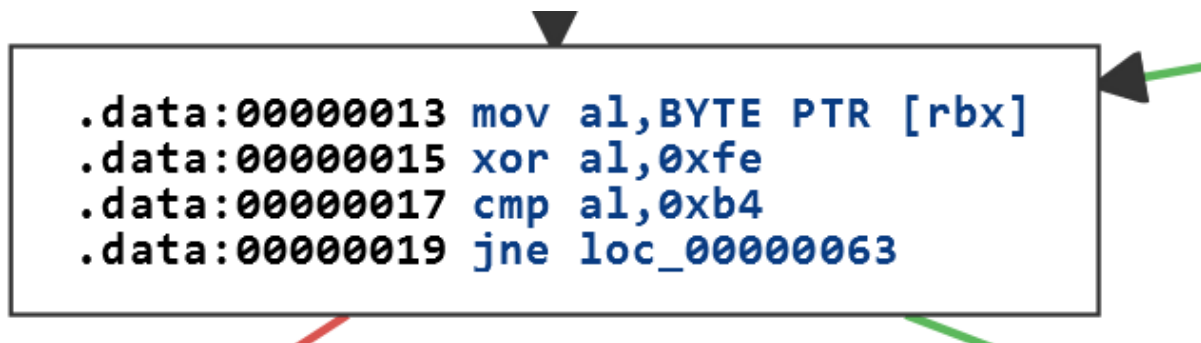


Figure 7: Loop.

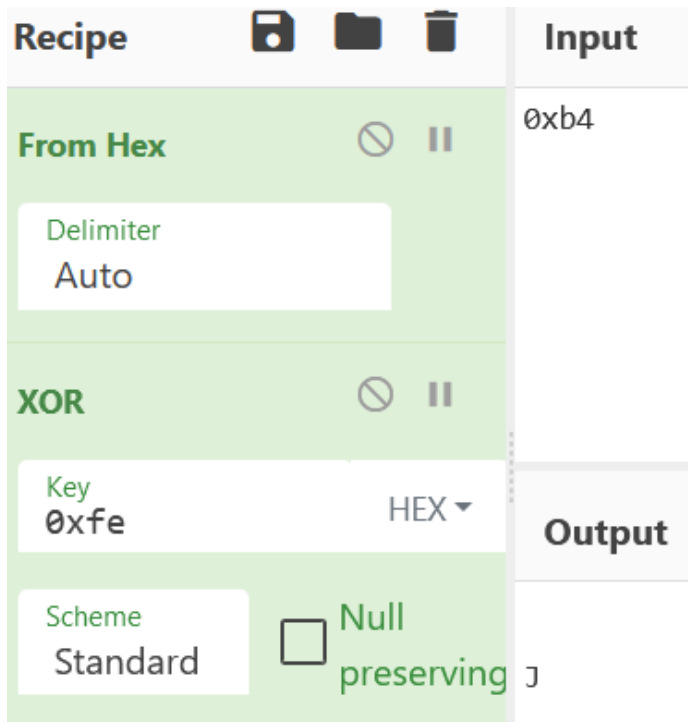


Figure 8: Single loops input XOR

Tested the possible password and it worked:

```
root@kali:~/Desktop/labs# ./lab06
Password: JAMKl0lz
correct!
root@kali:~/Desktop/labs#
```

Figure 9: Password test.

2. Time spent

Report:	1.5 h
Solving the lab:	7 h
Total:	8.5 h

