Reverse Engineering

Lab 05

Timo Lehosvuo, M3426

Report
Reverse Engineering, Marko Silokunnas
5.3.2021
ICT

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

Sisällys

1 Lab 05

Started the lab by going through the main function. There were a couple interesting functions like "_time", "_srand" and "_memset" function but these functions didn't lead me anywhere, so I jumped directly to the "check_serial" function:

```
mov     [esp+58h+var_58], 0
mov     [ebp+var_28], edx
call    _time
mov     [esp+58h+var_58], eax
call    _srand
lea     eax, aInsertSerialKe ; "Insert serial key: "
xor     ecx, ecx
mov     edx, 19
lea     esi_var23_ptr, [ebp+var_23]
mov     [esp+58h+var_58], esi_var23_ptr
mov     [esp+58h+var_54], 0
mov     [esp+58h+var_50], 19
mov     [ebp+var_2C], eax
mov     [ebp+var_30], ecx
mov     [ebp+var_34], edx
call    _memset
mov     eax, [ebp+var_2C]
mov     [esp+58h+var_58], eax
call    _printf
lea     ecx, aS          ; "%s"
lea     edx, [ebp+var_23]
mov     [esp+58h+var_58], ecx
mov     [esp+58h+var_54], edx
mov     [ebp+var_38], eax
call    ___isoc99_scanf
lea     ecx, [ebp+var_23]
mov     [esp+58h+var_58], ecx
mov     [ebp+var_3C], eax
call    check_serial
and     al, 1
mov     [ebp+var_24], al
test    [ebp+var_24], 1
```

Figure 1: Main function.

The "check_serial" function took some user input, made a copy of it, compared the length of it to "19" and it jumps to a loop depending on the result of the comparison:

```
var_C_userinput_1tavu= dword ptr -0Ch
var_8_userinput_kopio= dword ptr -8
var_1= byte ptr -1
arg_0_userinput= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 88          ; char *
mov     eax, [ebp+arg_0_userinput]
mov     [ebp+var_8_userinput_kopio], eax
mov     eax, [ebp+var_8_userinput_kopio]
mov     ecx, esp
mov     [ecx], eax
call    _strlen
cmp     eax, 19
jz      loc_80485CA
```

Figure 2: check_serial function.

The loops were the interesting part of the code and where I could eventually find the serial key:

```
loc_80485CA:
mov     eax, [ebp+var_8_userinput_kopio]
movsx   eax, byte ptr [eax]
mov     [ebp+var_C_userinput_1tavu], eax
call    ___ctype_b_loc
mov     eax, [eax]
mov     ecx, [ebp+var_C_userinput_1tavu]
movzx   eax, word ptr [eax+ecx*2]
and     eax, 2048
cmp     eax, 0
jnz     loc_80485F8
```

```
loc_80485F8:
mov     eax, [ebp+var_8_userinput_kopio]
movsx   eax, byte ptr [eax+1]
mov     [ebp+var_10], eax
call    ___ctype_b_loc
mov     eax, [eax]
mov     ecx, [ebp+var_10]
movzx   eax, word ptr [eax+ecx*2]
and     eax, 1024
cmp     eax, 0
jnz     loc_8048627
```

Figure 3: check_serial function loops.

Here the interesting parts are the byte number (e.g. user inputs 0th byte), "__ctype_b_loc" function and the numbers "2048" and "1024". The numbers stand for "Alphabetic" (1024) and "Numeric" (2048) these are related to ctype headers (https://grandidierite.github.io/looking-at-the-source-code-for-function-isalpha-isdigit-isalnum-isspace-islower-isupper-isxdigit-iscntrl-isprint-ispunct-isgraph-tolower-and-toupper-in-C-programming/). What the "__ctype_b_loc" function does is that it checks the inputs type and returns a pointer to a pointer. This pointer points to an array that contains characteristics of each single character. It is an internal function used by ctype headers. Basically what the loop does is it checks the userinput current bytes type and checks if it is numeric or alphabetic and then moves on or breaks. At this point I was really confused that how do I know the users inputs until I realised that this code only checks the type and not e.g. what number the user inputs. Looking at the loop you can see that it jumps from "eax+3" to "eax+5" and again at 8 and 13 so I thought that it might look like an actual serial code

“xxxx-xxxx-xxxx-xxxx”. The parts where it jumps the code accepts anything so I wrote the serial key down, tested it on kali and it worked:

```
serialkey.txt – M
Tiedosto  Muokkaa
0 number
1 letter
2 letter
3 number
-
5 number
6 number
7 letter
8 letter
-
10 letter
11 letter
12 number
13 number
-
15 number
16 number
17 number
18 number
```

Figure 4: Serial key input types.

```
root@kali:~/Desktop/labs# ./lab05
Insert serial key: 0kk0-00kk-kk00-0000
serial ok, starting game!
Guessing game!
Guess a number between 1-100: 50
too small!
Guess a number between 1-100: 70
too small!
Guess a number between 1-100: 80
too small!
Guess a number between 1-100: 90
too big!
Guess a number between 1-100: 85
too small!
Guess a number between 1-100: 87
too small!
Guess a number between 1-100: 89
too big!
Guess a number between 1-100: 88
you got it! it took you 7 guesses
root@kali:~/Desktop/labs#
```

Figure 5: Serial key test.

2. Time spent

Report:	1.5 h
Solving the lab:	4 h
Total:	5.5 h