jamk.fi

Reverse Engineering TTC6510-3002

Joonatan Ovaska

A K M MAHMUDUL HAQUE
AB0208

Student number: 2110841

Lab05
Date: 04.10.2023

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

First Step

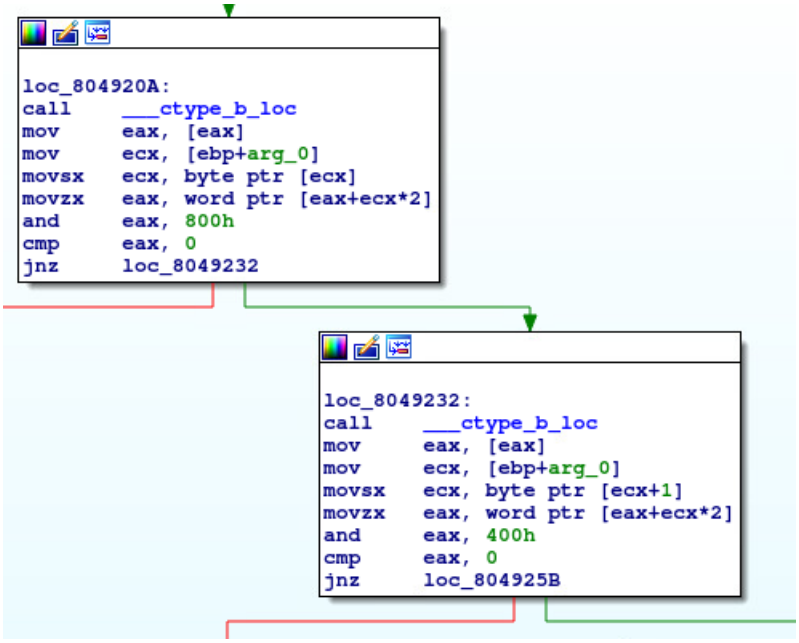
- **Game Initialization:** Main function orchestrates game-related processes once a valid serial key is provided.
- **Internal Functions:** Utilizes **_time** and **_srand** functions, although their specifics are not explored further as they are unrelated to the serial key validation.
- **User Input Handling:** Captures user input as a string using the **%S** format specifier with the **_scanf** function. This input likely serves as the serial key.

```
push    ebp
mov     ebp, esp
sub     esp, 48h
mov     eax, [ebp+argv]
mov     ecx, [ebp+argc]
xor     edx, edx
mov     [ebp+var_4], 0
mov     dword ptr [esp], 0
mov     [ebp+var_1C], eax
mov     [ebp+var_20], ecx
mov     [ebp+var_24], edx
call    _time
mov     [esp], eax
call    _srand
xor     eax, eax
lea     ecx, [ebp+var_17]
mov     [esp], ecx
mov     dword ptr [esp+4], 0
mov     dword ptr [esp+8], 13h
mov     [ebp+var_28], eax
call    _memset
lea     eax, aInsertSerialKe ; "Insert serial key: "
mov     [esp], eax
call    _printf
lea     ecx, [ebp+var_17]
lea     edx, aS          ; "%s"
mov     [esp], edx
mov     [esp+4], ecx
mov     [ebp+var_2C], eax
call    ___isoc99_scanf
lea     ecx, [ebp+var_17]
mov     [esp], ecx
mov     [ebp+var_30], eax
call    check_serial
and     al, 1
mov     [ebp+var_18], al
test    [ebp+var_18], 1
jz      loc_8049660
```

- **Serial Key Validation Process:** Program execution takes divergent paths based on the return value from the **check_serial** function.
- **Correct Serial Key:** If the serial key is correct, the **start_game** function is invoked. This function likely initializes the game environment and gameplay.
- **Incorrect Serial Key:** If the serial key is incorrect, the program prints a **"bad serial key, exiting…!"** message to the console. After displaying this message, the program exits.

Second Step

- **Overview of check_serial:**
  - User Input Processing: User input's memory address stored in the **eax** register.
  - The character size of the user input is determined using the **_strlen** function.
  - Size compared to 19, indicating a requirement of at least 19 characters for validation.

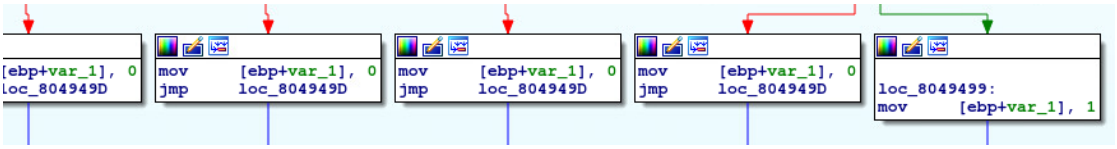


- **Characteristic Operations:**
  - 16 operations performed on the user input.
  - Each operation utilizes the **__ctype_b_loc** function to assess individual characters.
  - The **"and"** operation compares character characteristics to 0x800 and 0x400.
  - If any check fails, the program jumps out of the sequence and sets a variable (named **check_variable**) to zero.
  - **check_variable** is set to 1 only if all 16 steps are successful.
- **Serial Key Structure:**
  - The actual serial key comprises 16 characters.
  - Checking process involves indices [ecx], [ecx+1], [ecx+2], and [ecx+3] for the first set.
  - Subsequent sets skip every 4 checks: [ecx+5], [ecx+6], [ecx+7], [ecx+8], and so on until [ecx+18].
  - The pattern suggests the expected serial number format: XXXX-XXXX-XXXX-XXXX, where only the 'X' characters' characteristics are pertinent to the serial key validation.

Third Step

Check Algorithm and Exit Jumps:

- 16 exit jumps occur when a character fails the validation check.
- For instance, if the first five characters are acceptable but the sixth is not, the program immediately stops checking the remaining characters.
- At each incorrect character, the check algorithm exits, preventing further validation.



- **Correct Exit Jump:**
  - Only if all characters pass the validation checks does the correct exit jump occur.
  - When the entire serial key is validated successfully, the program takes the correct exit jump.
- **Controlled by check_variable:**
  - The program utilizes the **check_variable** to track the validation status.
  - If any character fails the check, **check_variable** is set to **0**, indicating an invalid serial key.
  - If all characters pass the checks, **check_variable** is set to **1**, signifying a valid serial key.

Step 4

Character Checks and "__ctype_b_loc" Function:

- **Endianness in Intel Assembly:**
  - Intel assembly operates on a little-endian system, a crucial detail for understanding the serial key algorithm.
  - Little-endian architecture stores the least significant byte of a multi-byte data at the lowest memory address.
  - Failure to consider endianness can lead to incorrect interpretations.

Understanding __ctype_b_loc Function:

- The **__ctype_b_loc** function is pivotal for character checks in the serial key algorithm.
- This function examines and characterizes each character, returning a value with specific bits set to **1**.
- The returned value from **__ctype_b_loc** carries essential information about character characteristics.
- These characteristics are fundamental for character validation in the serial key algorithm.

- Accurate interpretation of these characteristics is crucial for the algorithm's success.

Bit Index	Characteristic
0	Uppercase
1	Lowercase
2	Alphabetic
3	Numeric
4	Hexadecimal
5	Whitespace
6	Printing
7	Graphical
8	Blank
9	Characters (Control)
10	Punctuations
11	Alphanumeric

- The main check happens with and operation against 0x800 and 0x400.
- The expected characteristics of the key characters should be in hexadecimal format. However, the notion of including whitespace as a characteristic does not seem logical or meaningful.

Bit number	Value	Description
7	1	graphical
6	1	printing
5	0	whitespace
4	1	hexadecimal numeric
3	0	numeric
2	1	alphabetic
1	0	lowercase
0	1	uppercase
	0	
	0	
	0	
	0	
11	1	alphanumeric
10	0	punctuation
9	0	control
8	0	blank

https://braincoke.fr/blog/2018/05/what-is-ctype-b-loc/#reading-an-entry-of-__ctype_b_loc

- When the binary little-endian 0x800 and 0x400 indexed as shown in Figure 4.

  It reveals the needed characteristics of the serial key.

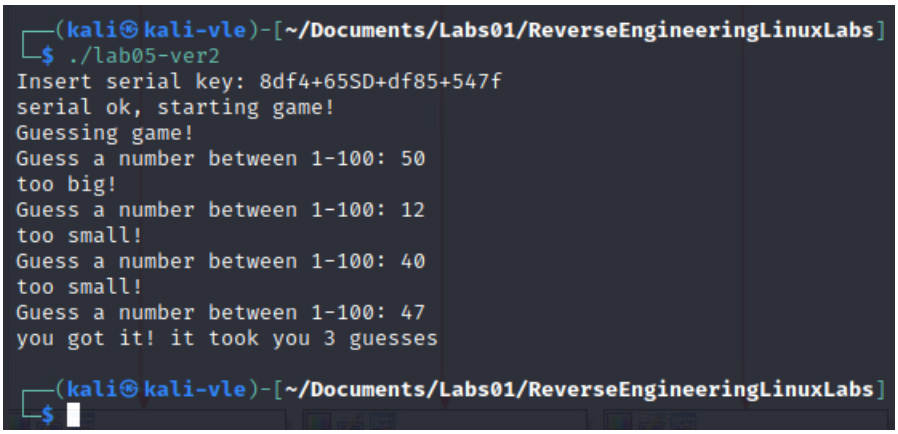
Bit index	7	6	5	4	3	2	1	0					11	10	9	8
0x800	0	0	0	0	1	0	0	0	0	0	0	0	0	0		0
0x400	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

- Considering this information, it's evident that the anticipated serial key comprises both Numeric (0x800) and Alphabetic (0x400) characters. The next step involves identifying the specific characters that fall into these categories. Examining the algorithm reveals the characters subjected to the and operation with 0x800 and 0x400, clarifying which ones are considered numeric and alphabetic.

Index	**AND operation**	**Characteristics**
0	0x800	**N**umeric
1	0x400	**A**lphabetic
2	0x400	**A**lphabetic
3	0x800	**N**umeric
4		
5	0x800	**N**umeric
6	0x800	**N**umeric
7	0x400	**A**lphabetic
8	0x400	**A**lphabetic
9		
10	0x400	**A**lphabetic
11	0x400	**A**lphabetic
12	0x800	**N**umeric
13	0x800	**N**umeric
14		
15	0x800	**N**umeric
16	0x800	**N**umeric
17	0x800	**N**umeric
18	0x400	**A**lphabetic

Serial keys

Here is one acceptable serial key: **8df4+65SD+df85+547f**



Topic	Time
Lab05	15 hours
Report writing	3 hours

Note: I had to take a lot of help from classmates.