jamk.fi

Reverse Engineering TTC6510-3002

Joonatan Ovaska

A K M MAHMUDUL HAQUE
AB0208

Student number: 2110841

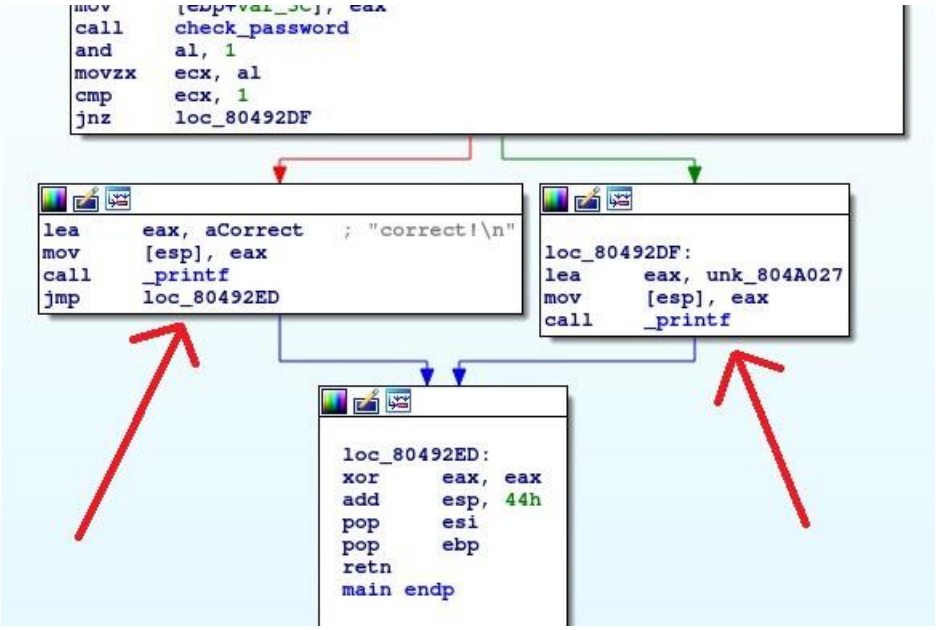
Lab04
Date: 19.09.2023

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

First Step

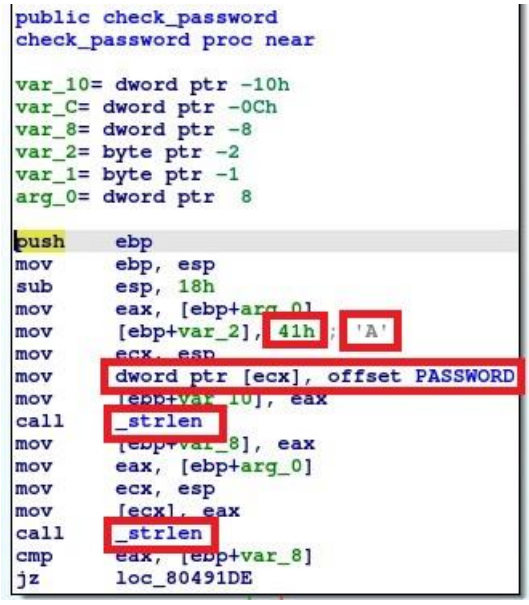
```
push    ebp
mov     ebp, esp
push    esi
sub     esp, 44h
mov     eax, [ebp+argv]
mov     ecx, [ebp+argc]
xor     edx, edx
mov     [ebp+var_8], 0
lea     esi, [ebp+var_26]
mov     [esp], esi
mov     dword ptr [esp+4], 0
mov     dword ptr [esp+8], 1Eh
mov     [ebp+var_2C], eax
mov     [ebp+var_30], ecx
mov     [ebp+var_34], edx
call    _memset
lea     eax, aPassword  ; "Password: "
mov     [esp], eax
call    _printf
lea     ecx, [ebp+var_26]
lea     edx, aS         ; "%s"
mov     [esp], edx
mov     [esp+4], ecx
mov     [ebp+var_38], eax
call    ___isoc99_scanf
lea     ecx, [ebp+var_26]
mov     [esp], ecx
mov     [ebp+var_3C], eax
call    check_password
and     al, 1
movzx   ecx, al
cmp     ecx, 1
jnz     loc_80492DF
```

- The code gets ready to do some work by setting up a space for storing information and variables.
- It asks the user to enter a password by showing **"Password: "**.
- When the user types in the password and hits Enter, the code takes that input and remembers it as a series of characters (like a word).
- It then sends this entered password to a function called **check_password** to see if it's the right one.
- Before making the final decision, the code plays around with some bits. It looks at the lowest bit in the entered password and checks if it's set to 1.
- If that bit is indeed 1, it continues; otherwise, it goes somewhere else (to **"loc_80492DF"**).



- Next, the code checks what the **check_password** function says:
  - If the function says "Yes, that's the right password" (returns 1), the code celebrates by saying **"correct!\n"** and finishes its job.
  - If the function says "No, that's not the right password" (returns 0), the code takes a different path.

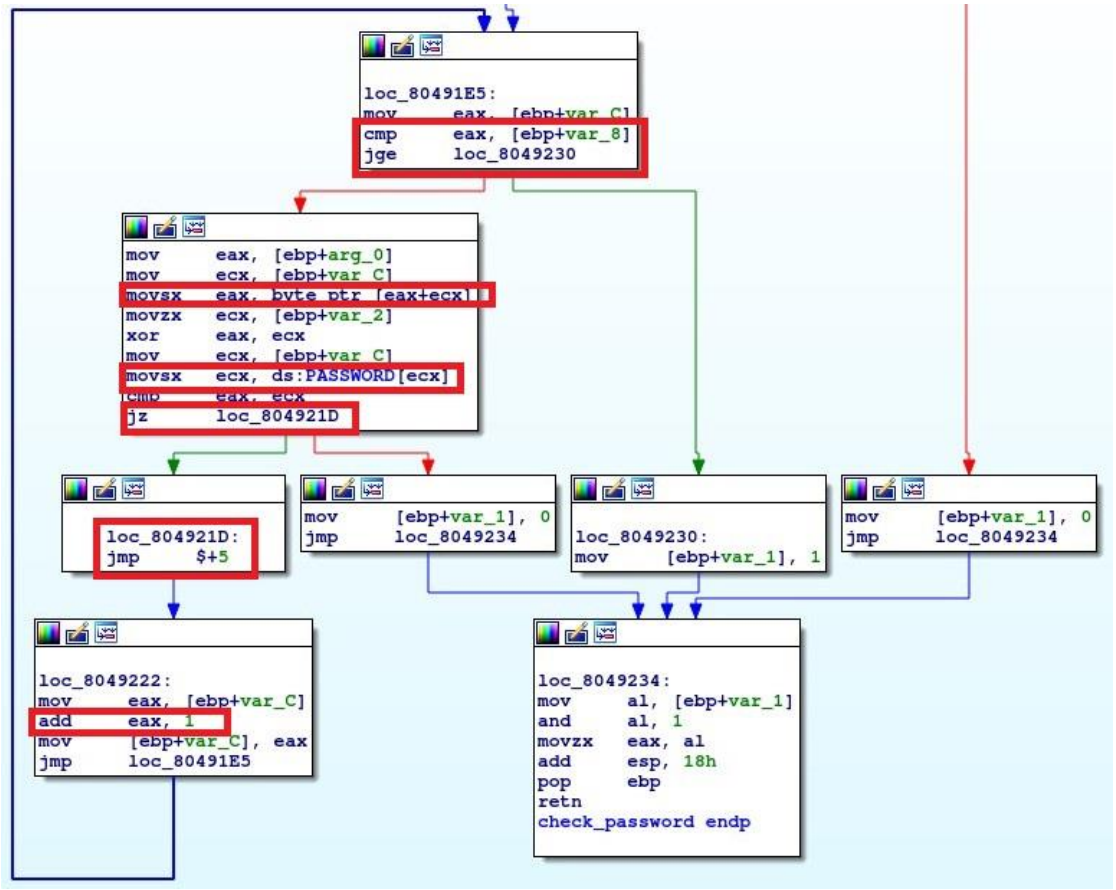
Second Step



- First, the function figures out how long the user's input password is.
- It sets up an encryption key with the letter 'A'.
- Then, it checks if the length of the entered password matches the length of a predefined password (PASSWORD) in two ways:
  - First, it goes character by character, comparing the lengths.

- Second, it compares the actual characters one by one. This keeps happening until all the characters are compared.

- If the lengths match, it means there's a chance the password is right. It checks this by comparing the user's input character by character with the **PASSWORD.**



- In the loop, a counter keeps track of the position in the input.
- The loop does this:
  - There's this loop counter thing, like keeping track of which step we're on.
  - We grab one character at a time from our message and make it look like a 32-bit secret code.
  - If it's a positive character, we add lots of zeros to the left (like 00000101).
  - If it's a negative character, we add ones instead (like 11111010).
  - We also have a secret key (**XOR key**) hidden away. It's like a secret sauce for our operation.
  - **Now, the magic happens:** we mix our character with the secret sauce using **XOR**.
  - The result becomes a new secret code, which we keep in a safe spot (**eax**).
  - Next, we compare this secret code with a character from a super-secret **PASSWORD**.
  - This check tells us if we're on the right track. It's like confirming we're using the right clue.
- The program's flow depends on these comparisons and jumps around accordingly.

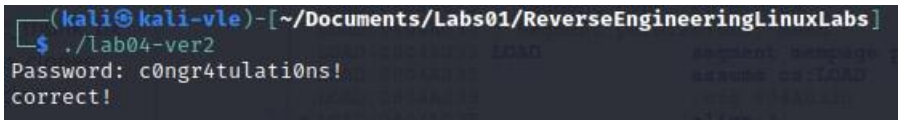
Password

```
.rodata:0804A008                public PASSWORD
.rodata:0804A008 PASSWORD       db 22h                  ; DATA XREF: ch
.rodata:0804A008                                        ; check_passwor
.rodata:0804A009 aQ3u545Q2      db 'q/&3u54- 5(q/2`',0
.rodata:0804A019 aPassword      db 'Password: ',0       ; DATA XREF: ma
.rodata:0804A024 aS             db '%s',0               ; DATA XREF: ma
```

- The encrypted password we have is **"q/&3u54- 5(q/2`"**, and since there are 16 characters, it suggests the original password is also 16 characters long.
- The encryption is done using a simple **XOR** operation, which is like a secret code. The key for this operation is set to 'A', which has an ASCII code of **0x41**.
- The address of the real password (the one we want to find) is stored in a special spot called the **"ecx register".** Think of it like having a map to the treasure.
- With the **XOR** operation, it looks like the original password (PASSWORD) was encrypted. It's like taking a message and jumbling it up with a secret pattern.
- To decrypt the encrypted string, we need a key, which is indicated in the code as the hex value 41 (0x41), which corresponds to 'A' in ASCII.
- The XOR cipher is used to modify the bits of the encrypted string. It's like having a secret decoder ring.
- If we use the XOR decryption method, we can crack the password. Think of it like using a special tool to unscramble the secret message.

- The decrypted plaintext is **0ngr4tulati0ns!.** But this password was incorrect. It was clear something was missing.
- The solution could be found by returning to the password **rodata** viewable by clicking the PASSWORD string. The **db** (define byte) directive is used to define a byte in memory. The value **22h** is specified as the content of this byte.

- In the code, I see the value 0x22, which is the hexadecimal representation of the ASCII character for a double-quotation mark **(")**.
- This line of code essentially sets the PASSWORD variable in memory to hold the double-quotation mark character.
- So, when the program looks at the PASSWORD variable in memory, it sees the double-quotation mark character, which suggests that the complete encrypted password is indeed **"q/&3u54- 5(q/2`".**
- When I decrypt the encrypted string using the key 0x41 (which is 'A' in ASCII), the correct password was revealed.



- I can confirm this by running the program in a terminal.

Topic	**Time**
Lab04	7 hours
Report writing	3 hours