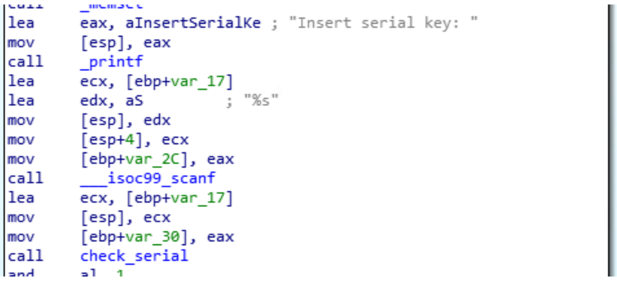
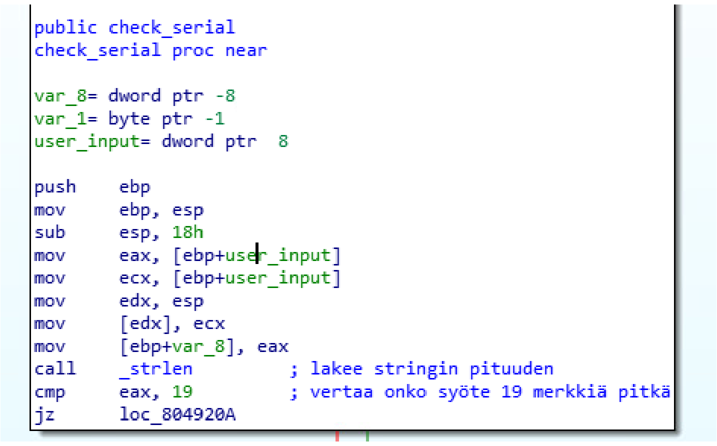
Kalle Jalkanen AB8822

In main there is ”Insert serial key: “ witch we see when we launch program. It waits input to be a string because of “%s”. Next main thing here is when it calls ‘check_serial’.



```
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
```

In next part program checks input length and compares it to 19. If input maches it jumps to next checking part.

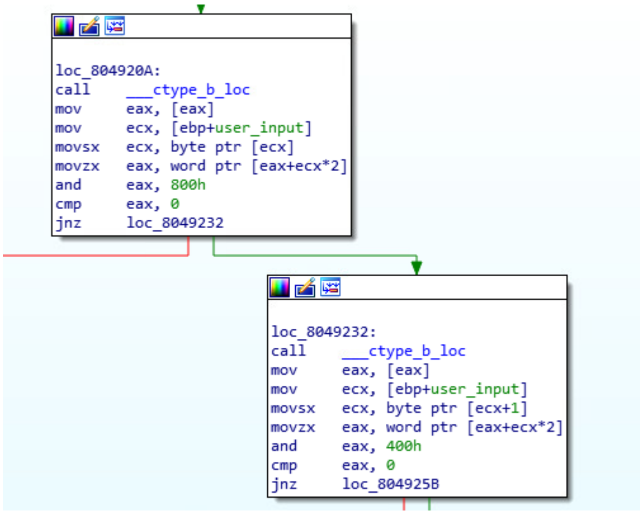


```
public check_serial
check_serial proc near

var_8= dword ptr -8
var_1= byte ptr -1
user_input= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     eax, [ebp+user_input]
mov     ecx, [ebp+user_input]
mov     edx, esp
mov     [edx], ecx
mov     [ebp+var_8], eax
call    _strlen         ; lakee stringin pituuden
cmp     eax, 19         ; vertaa onko syöte 19 merkkiä pitkä
jz      loc_804920A
```

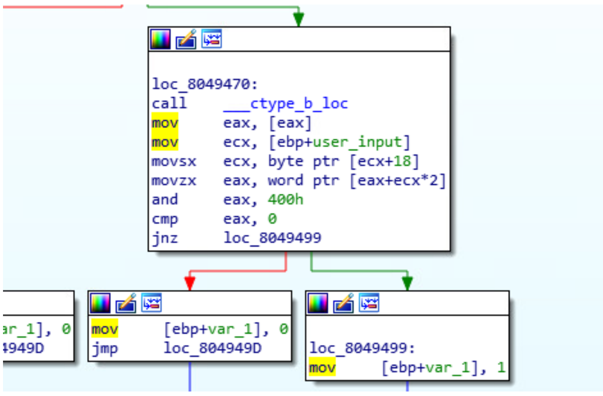
Next there is 16 different boxes. __ctype_b_loc is some kind of character checker. It checks is input character, number, space and so on. It looks like there is running number, it starts with esx and at every box it rises like ecx+1, ecx+2 and so on. But it jumps over every fifth number. In every box it checks if input character matches and if it is correct it jumps to next check. There looks like to be hex number 800h or 400h in every box.



```
loc_804920A:
call    ___ctype_b_loc
mov     eax, [eax]
mov     ecx, [ebp+user_input]
movsx   ecx, byte ptr [ecx]
movzx   eax, word ptr [eax+ecx*2]
and     eax, 800h
cmp     eax, 0
jnz     loc_8049232
```

```
loc_8049232:
call    ___ctype_b_loc
mov     eax, [eax]
mov     ecx, [ebp+user_input]
movsx   ecx, byte ptr [ecx+1]
movzx   eax, word ptr [eax+ecx*2]
and     eax, 400h
cmp     eax, 0
jnz     loc_804925B
```

It only gives [ebp+var_1] value of 1 if every check passes, otherwise it gives value of 0.



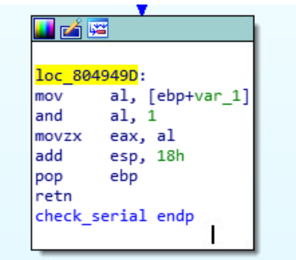
```
loc_8049470:
call    ___ctype_b_loc
mov     eax, [eax]
mov     ecx, [ebp+user_input]
movsx   ecx, byte ptr [ecx+18]
movzx   eax, word ptr [eax+ecx*2]
and     eax, 400h
cmp     eax, 0
jnz     loc_8049499
```

```
ar_1], 0
4949D
```

```
mov     [ebp+var_1], 0
jmp     loc_804949D
```

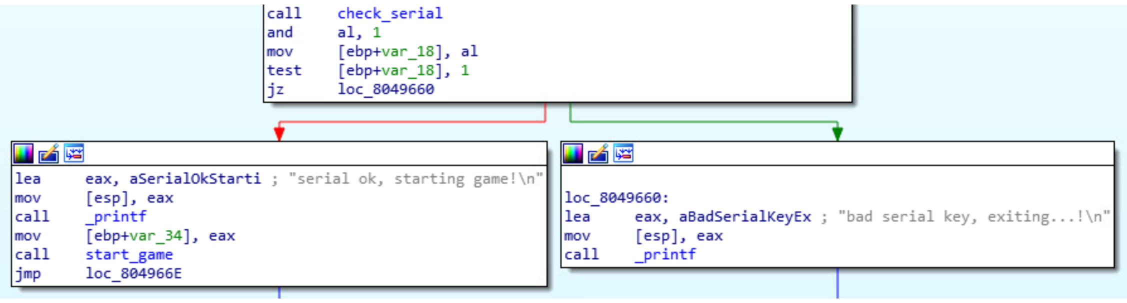
```
loc_8049499:
mov     [ebp+var_1], 1
```

In next part it stores that var_1 to al. Then it does and operation to it and stores al's one byte value to eax.



```
loc_804949D:
mov     al, [ebp+var_1]
and     al, 1
movzx   eax, al
add     esp, 18h
pop     ebp
retn
check_serial endp
```

When it returns to main it does and operation for al and stores it to [ebp+var_18]. Then it does test operation to it. 'test' is same kind of operator than 'and' but it does not store the answer anywhere.

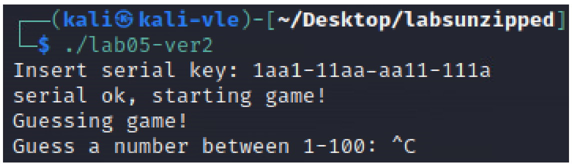


```
call    check_serial
and     al, 1
mov     [ebp+var_18], al
test    [ebp+var_18], 1
jz      loc_8049660
```

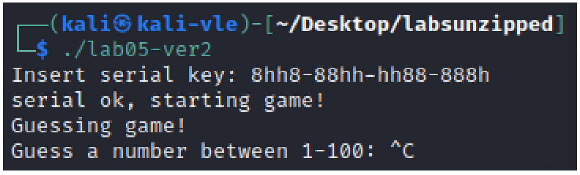
```
lea     eax, aSerialOkStarti ; "serial ok, starting game!\n"
mov     [esp], eax
call    _printf
mov     [ebp+var_34], eax
call    start_game
jmp     loc_804966E
```

```
loc_8049660:
lea     eax, aBadSerialKeyEx ; "bad serial key, exiting...!\n"
mov     [esp], eax
call    _printf
```

I started to test serial keys different ways and in the end I realized that 800h is number and 400h is letter. Every fifth character witch it jumps over in boxes is -. So, there is no one right answer to this it only checks if number and letters are in right order.



```
┌──(kali㉿kali-vle)-[~/Desktop/labsunzipped]
└─$ ./lab05-ver2
Insert serial key: 1aa1-11aa-aa11-111a
serial ok, starting game!
Guessing game!
Guess a number between 1-100: ^C
```



```
┌──(kali㉿kali-vle)-[~/Desktop/labsunzipped]
└─$ ./lab05-ver2
Insert serial key: 8hh8-88hh-hh88-888h
serial ok, starting game!
Guessing game!
Guess a number between 1-100: ^C
```

Time used 3,5 hours.