

Reverse Engineering

Final Report

Shannon Dsouza
11-12-2020

Index

Synthesis.....	2
Analyzing the program in a disassembler.....	3
Executing the program in a debugger.....	9
Patching the program.....	20

Synthesis

Reverse engineering the application allows us to work out the serial to unlock the application. Alternatively, we can also edit the opcodes to patch the program and unlock it without the correct serial.

A disassembler is used to reveal the flow of the program and identify the functions in the program.

A debugger is used to view the registers at each step of execution of the program. The values at the registers can be seen and modified at any point during the execution. With this information, we reverse engineer the operations performed to identify the correct serial.

The program is patched by editing the opcodes in a hexeditor, changing the flow of the program and unlocking it without the correct serial.

Reverse Engineering Process

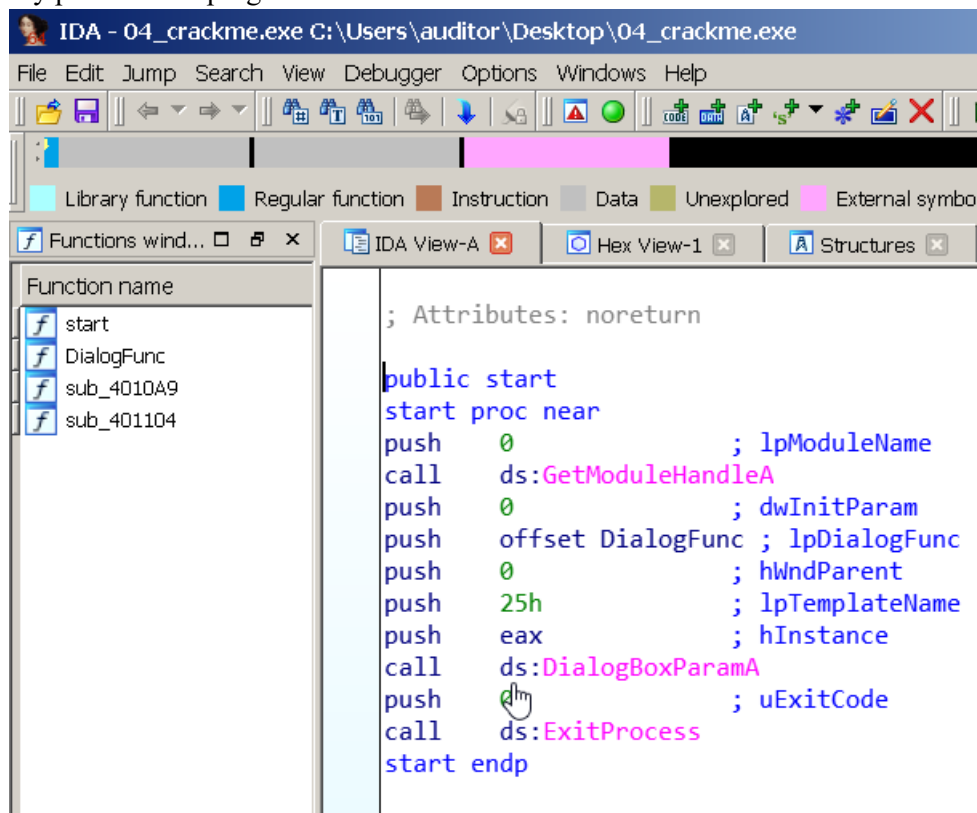
Analyzing the program in a disassembler

Description:

Using a disassembler, we are able to see the machine code in blocks. Using this graph view we see exactly which commands execute and their order and try to work out (reverse engineer) the correct values that must be in the registers to form the expected serial.

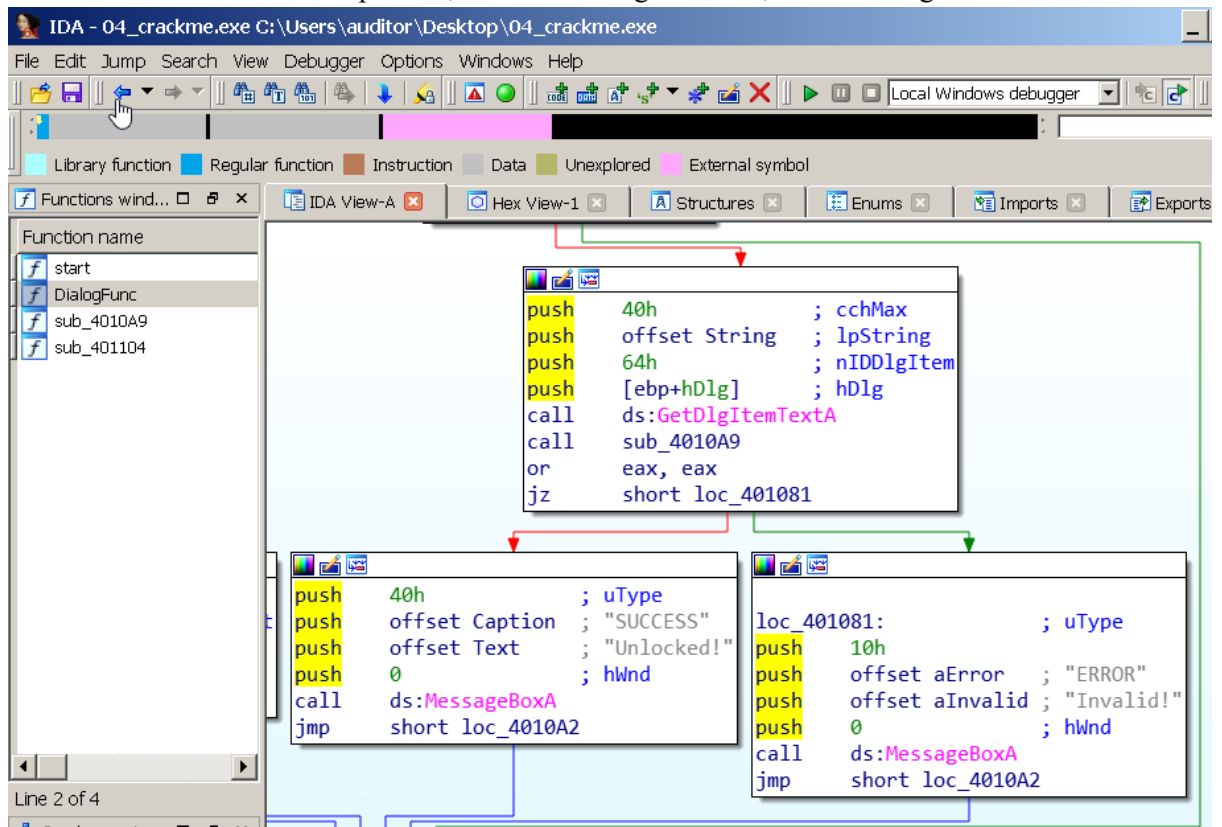
Process:

- a) Opening the application in 'The Interactive Disassembler'^[1] (IDA) by Hex-Rays SA displays the entry point for the program.



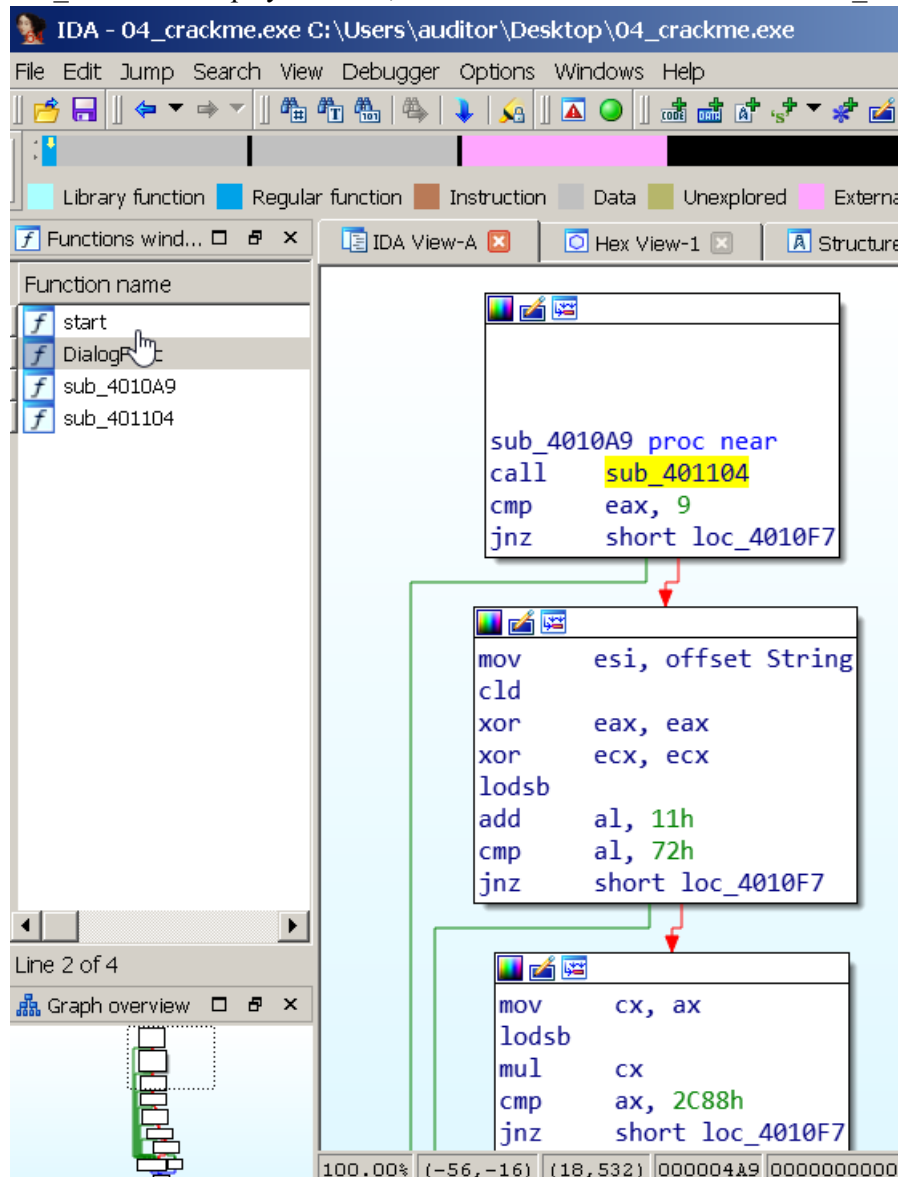
On the left tab the identified functions are displayed

- b) Opening the DialogFunc function from the left tab opens the graph view of all the dialog box functions such as the Serial input box, Success message box and, Error message box.



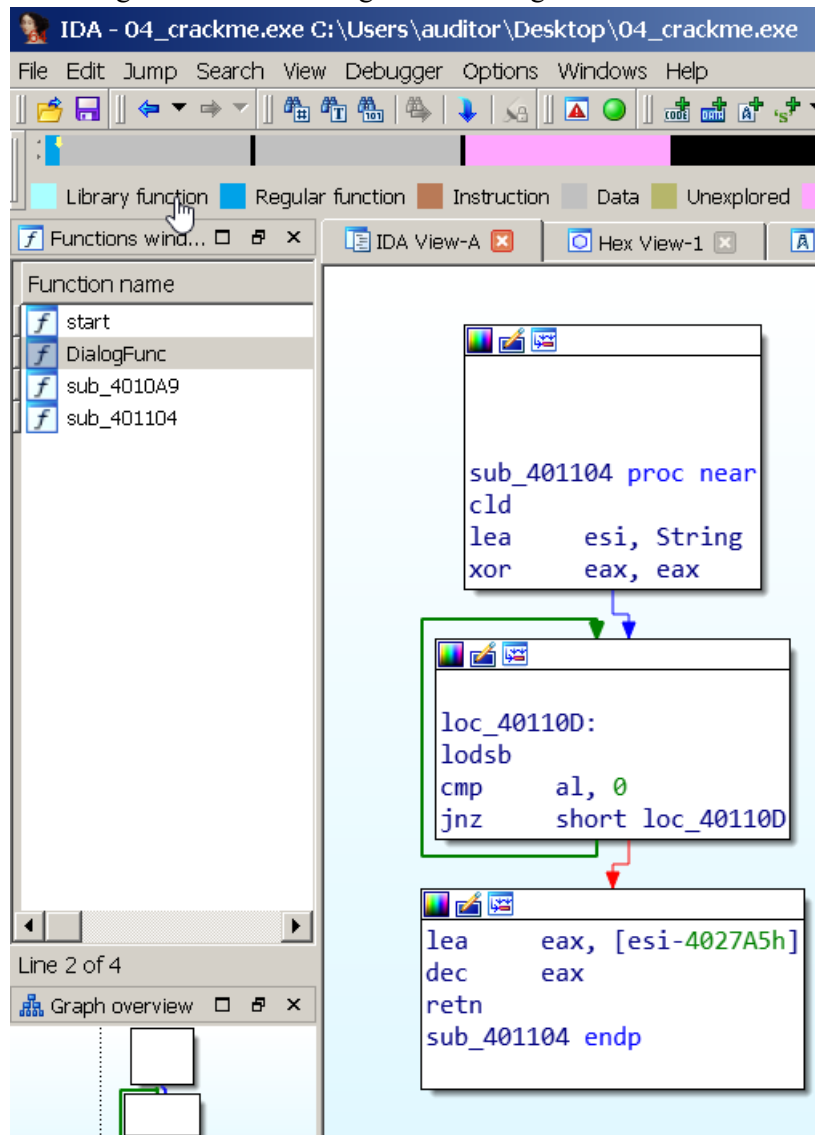
The GetDlgItemTextA is the input string (Serial) function, the next call is to function:
sub_4010A9

c) Function sub_4010A9 is displayed below, the first action is a call to function sub_401104.



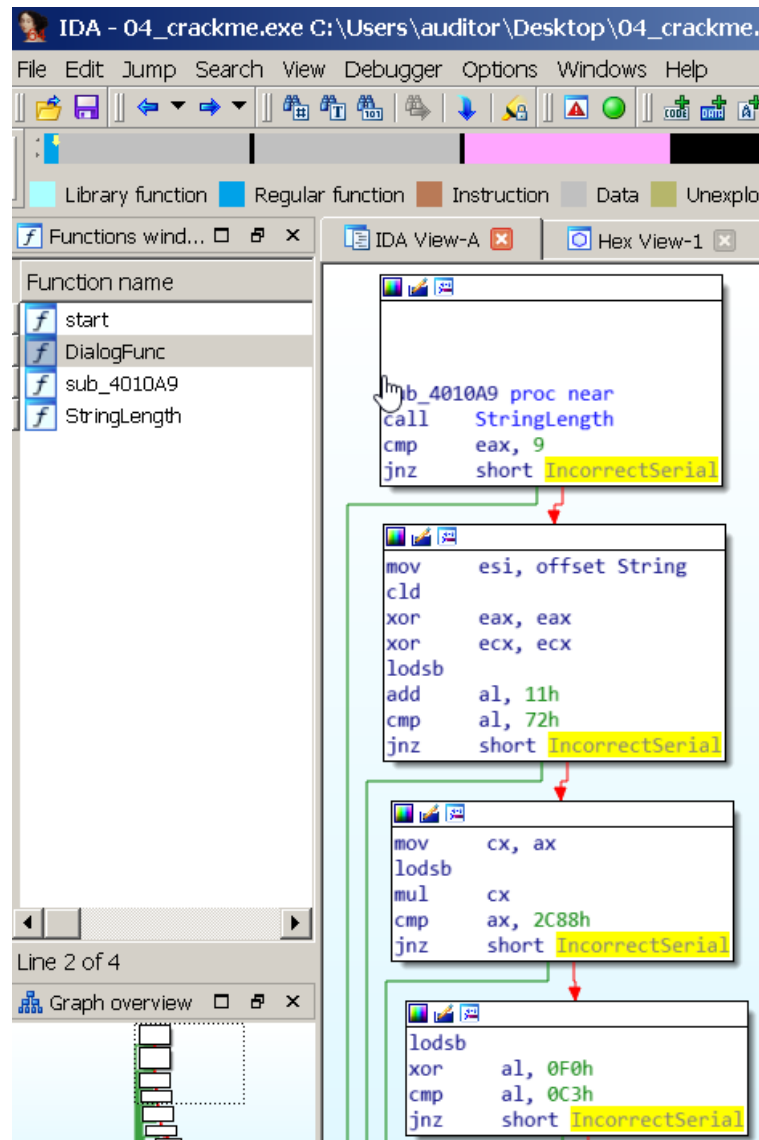
This function calculates the length of the string

- d) The function sub_401104 is a function that calculates the length of the user inputted string. The address of the string is loaded into ESI and then updated as each character is loaded into the AL register, finally the address (value) of ESI is subtracted from the address of the beginning of the string to calculate the length of the string.

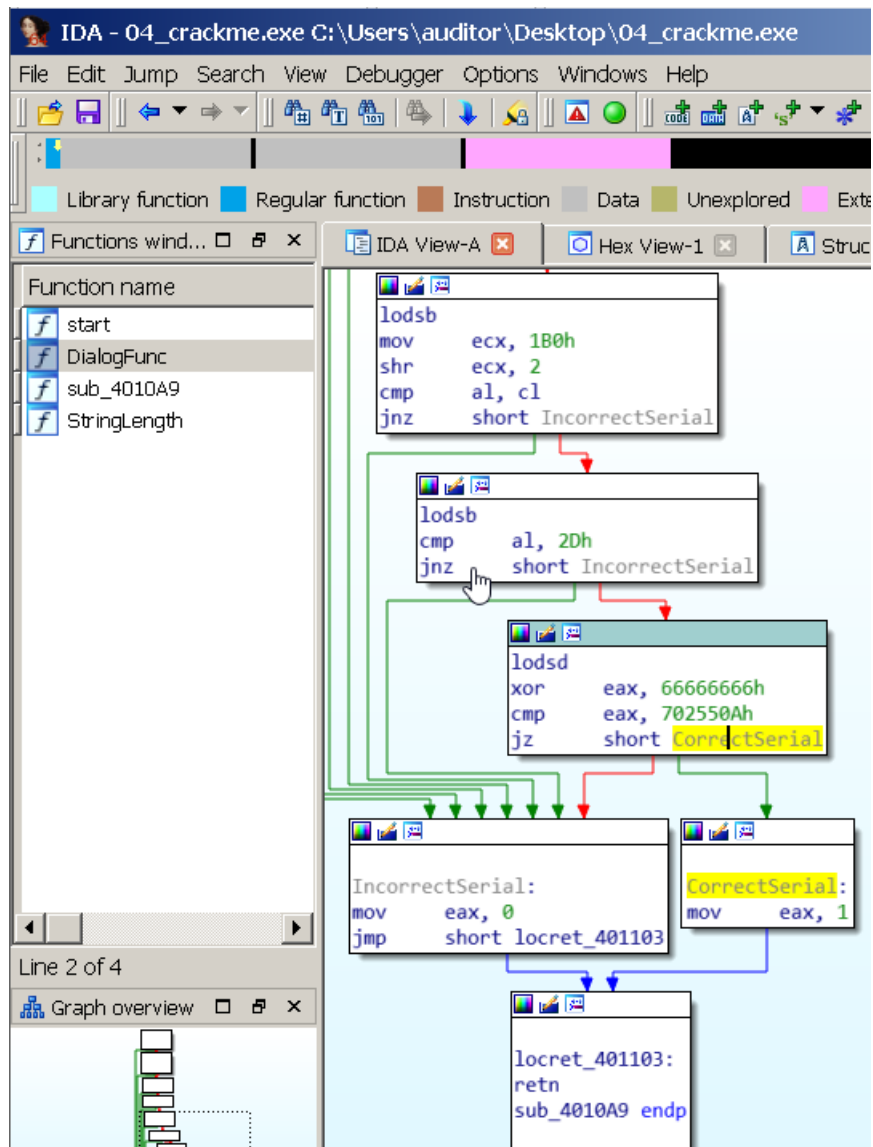


The function returns the calculated length

- e) Returning to the previous function sub_4010A9, rename the functions for the sake of clarity. The value returned from the function that calculates the string length is compared to the value 9, therefore the correct serial is 9 characters long. The rest of the flow of the program can be followed here, we can use a debugger to see the values in every register at each step of execution.



Flow of execution (I)



Flow of execution (II)

Reference:

^[1]<https://www.hex-rays.com/products/ida/>

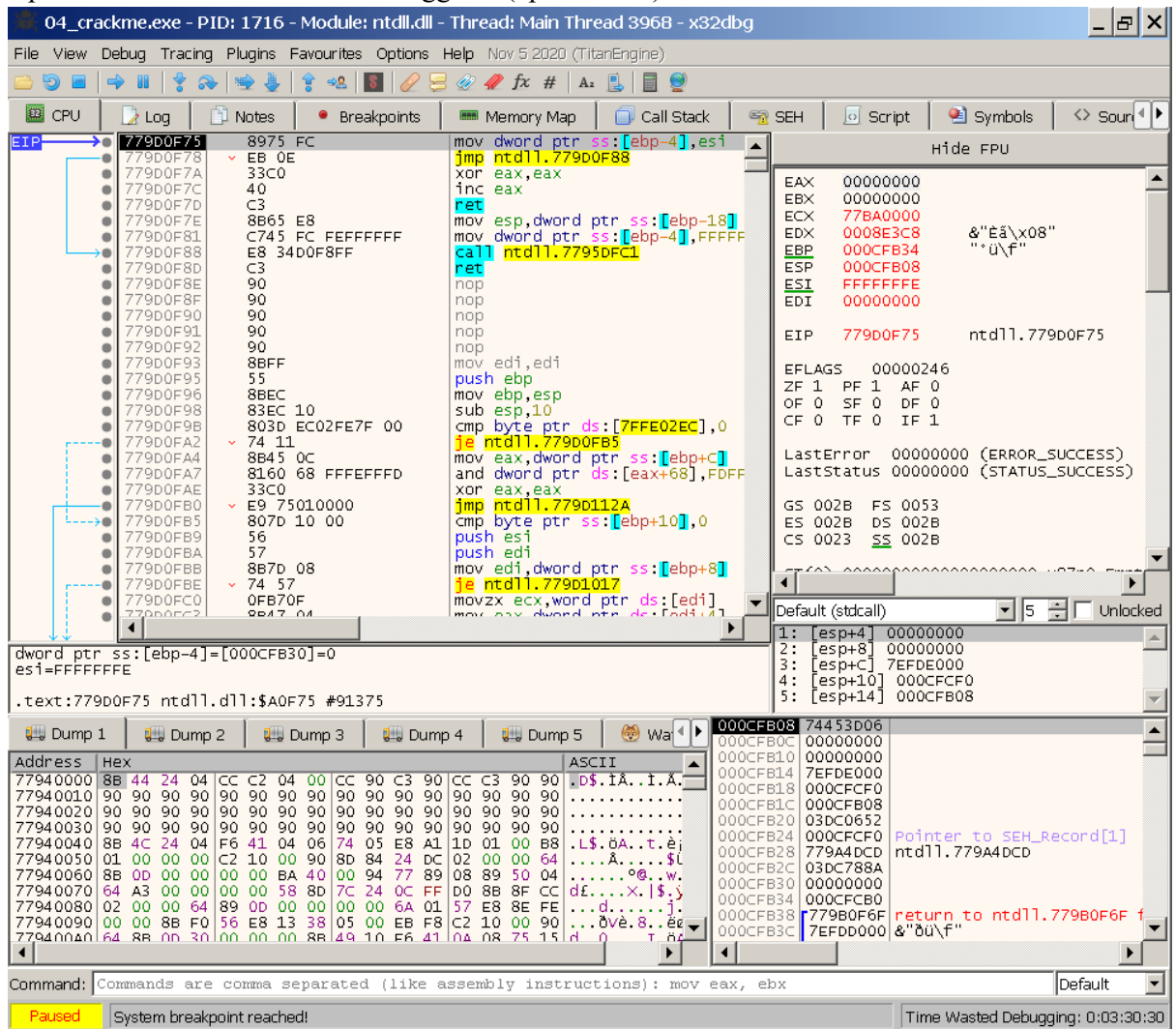
Executing the program in a debugger

Description:

Using a debugger allows us to see the values in each of the registers at each step of the execution of the program. This process allows us to calculate what each character of the serial must be by using the operation and the expected result (using cmp).

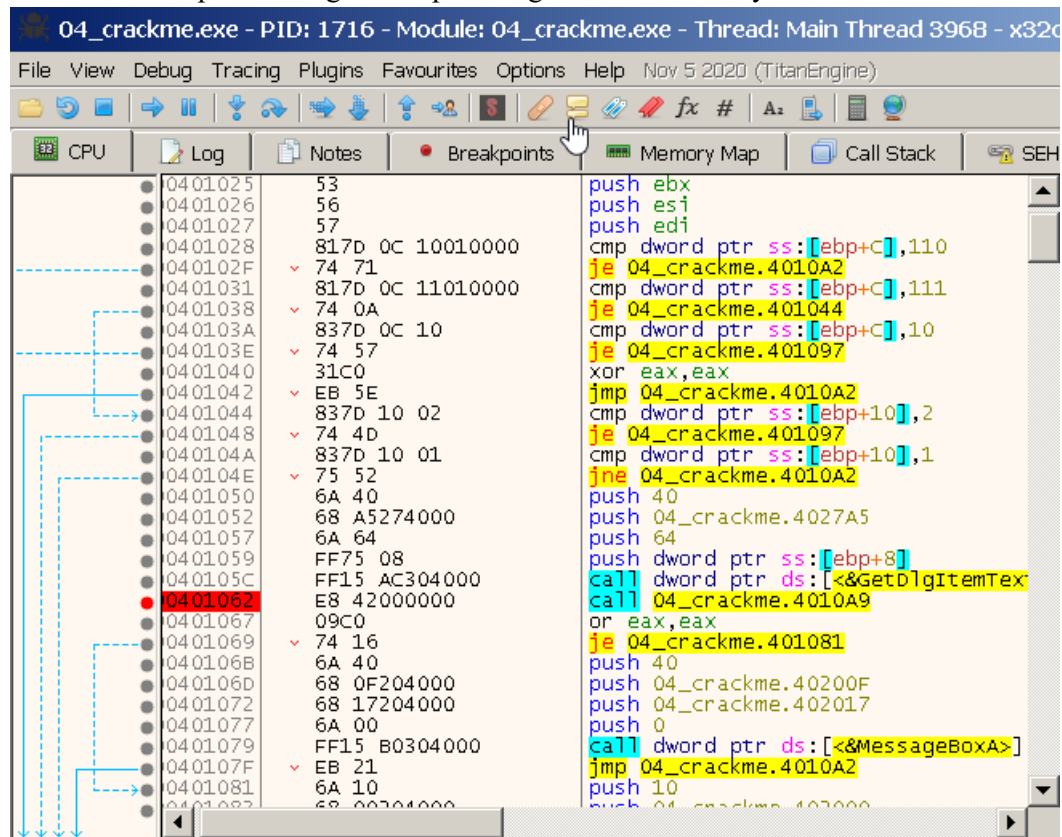
Process:

- Open the executable in the x64 debugger^[2] (open source).



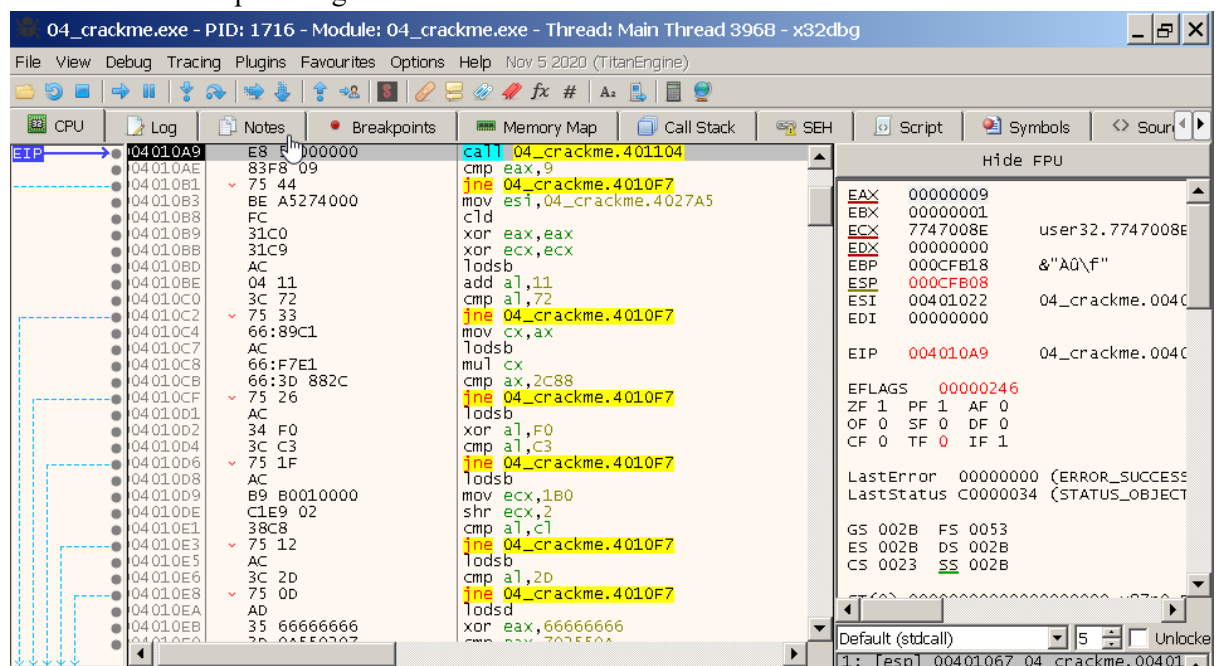
The module that is opened first is the ntdll.dll

- b) Clicking 'run' brings us to the next module, the executable we're trying to reverse engineer. The <GetDlgItemTextA> call is the input string as noted from the disassembler analysis. By adding a breakpoint in the step after the input string we can pause the program execution as soon as the user inputs a string. The input string used for this analysis is 'aaaaaaaa'.



The program hits the breakpoint and pauses execution after the user inputs a string

- c) Using the 'step into' button we enter the function 4010A9. This is the function that verifies whether the user input string is the correct serial.



The first function 401104 is the function that calculates the string length. It's known that the expected string length is 9 so we 'step over' that function.

- d) The string length function returns the length in the EAX register. As the string length is 9 (EAX=9) in this case, the jump to 4010F7 (incorrect serial) is not taken.

04_crackme.exe - PID: 1716 - Module: 04_crackme.exe - Thread: Main Thread 3968 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Nov 5 2020 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

Hide FPU

Address	Disassembly	Comment
004010A9	E8 56000000	call 00401000
004010AE	83F8 09	cmp eax, 9
004010B1	75 44	jne 004010F7
004010B3	BE A5274000	mov esi, 004010F7
004010B8	FC	cld
004010B9	31C0	xor eax, eax
004010BB	31C9	xor ecx, ecx
004010BD	AC	lodsb
004010BE	04 11	add al, 11
004010C0	3C 72	cmp al, 72
004010C2	75 33	jne 004010F7
004010C4	66 89C1	mov cx, ax
004010C7	AC	lodsb
004010C8	66 F7E1	mul cx
004010CB	66 3D 882C	cmp ax, 2C88
004010CF	75 26	jne 004010F7
004010D1	AC	lodsb
004010D2	34 F0	xor al, F0
004010D4	3C C3	cmp al, C3
004010D6	75 1F	jne 004010F7
004010D8	AC	lodsb
004010D9	B9 B0010000	mov ecx, 1B0
004010DE	C1E9 02	shr ecx, 2
004010E1	38C8	cmp al, cl
004010E3	75 12	jne 004010F7
004010E5	AC	lodsb
004010E6	3C 2D	cmp al, 2D
004010E8	75 0D	jne 004010F7
004010EA	AD	lodsd
004010EB	35 66666666	xor eax, 66666666
004010ED	35 00000000	cmp eax, 00000000

Jump is not taken
004010F7
.text:004010B1 04_crackme.exe:\$10B1 #4B1

EAX 00000009
EBX 00000001
ECX 7747008E user32.7747008E
EDX 00000000
EBP 000CFB18 &"A0\ff"
ESP 000CFB08
ESI 004027AF 04_crackme.004027AF
EDI 00000000
EIP 004010B1 04_crackme.004010B1

EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_EXISTS)

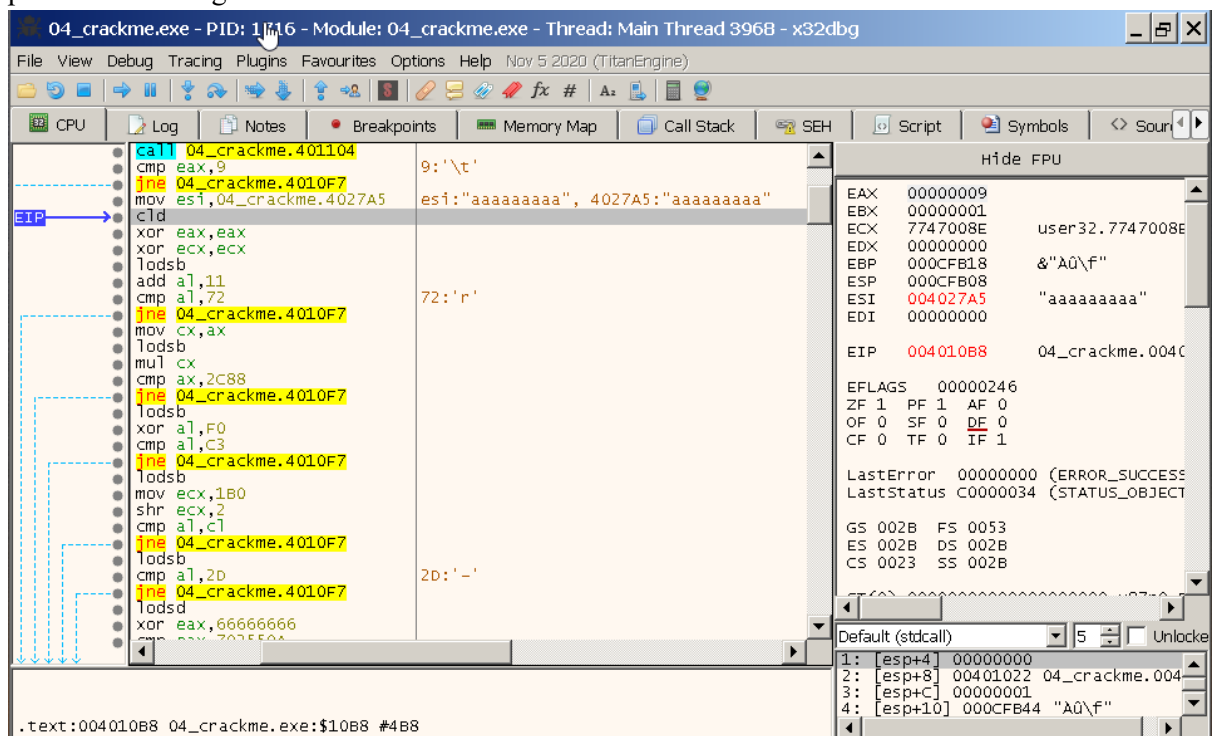
GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B

Default (stdcall) 5 Unlocked

1: [esp+4] 00000000
2: [esp+8] 00401022 04_crackme.00401022
3: [esp+C] 00000001
4: [esp+10] 000CFB44 "A0\ff"

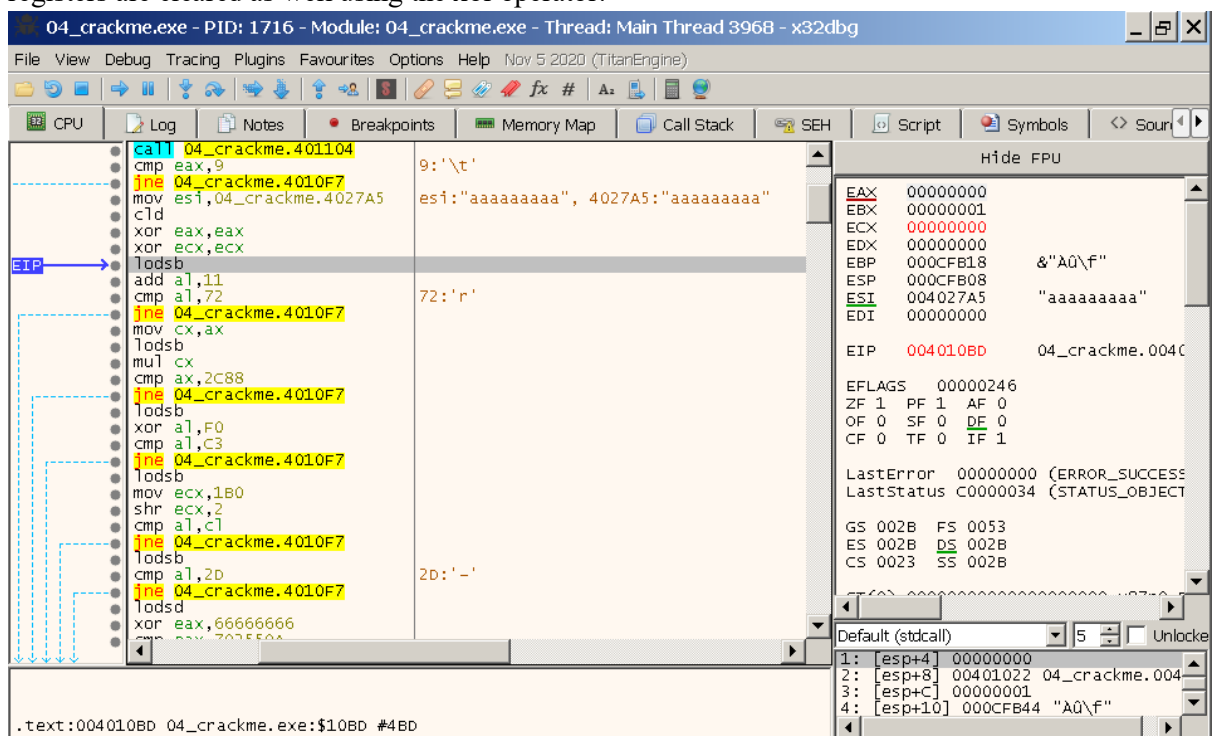
The jump is not taken

- e) In the next step, the location of the string: 4027A5 is moved into the ESI register. ESI now points at the string 'aaaaaaaa'.



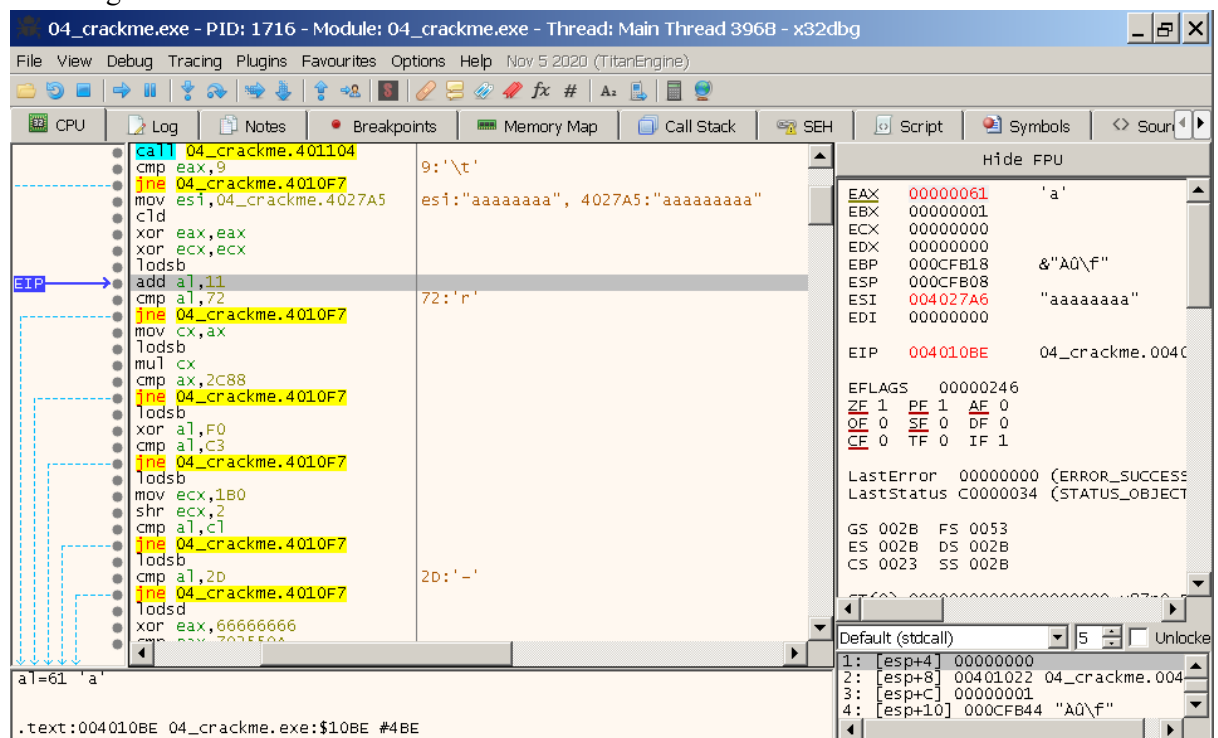
The ESI register now has the value 4027A5 which is the location of the user inputted string

- f) The next three steps clear the direction flag, which is already set to 0. The EAX and ECX registers are cleared as well using the xor operator.



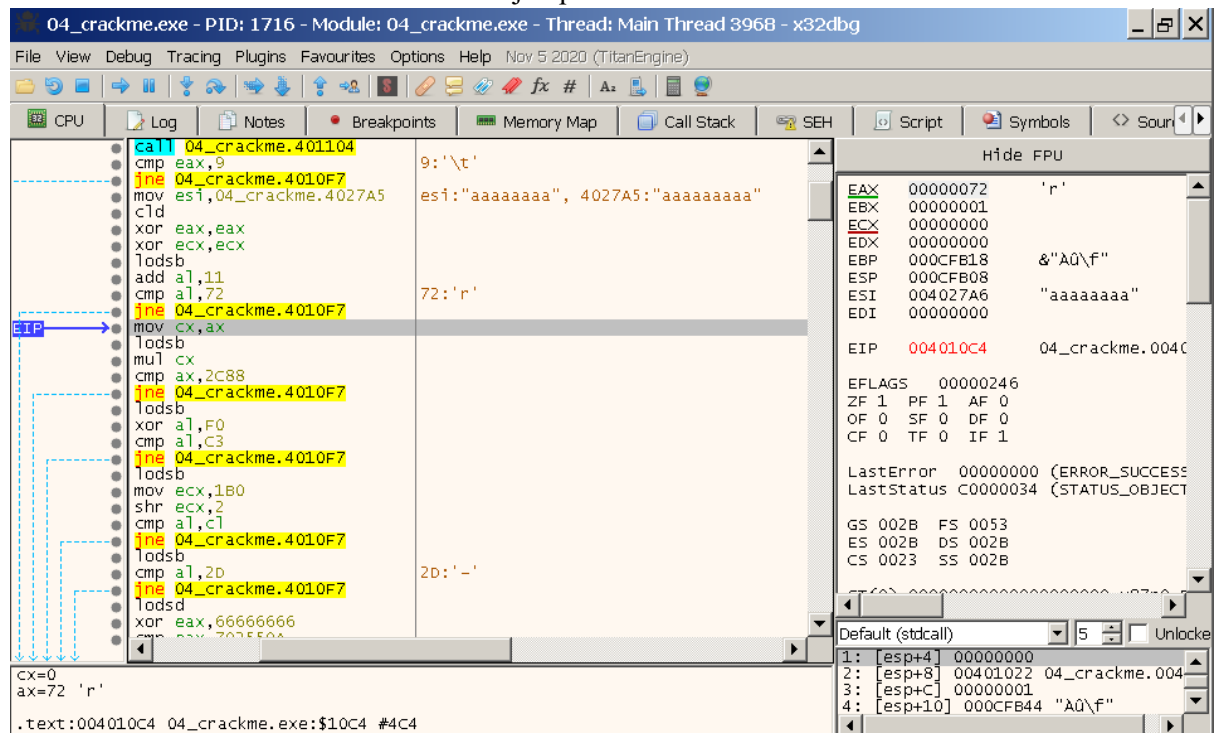
EAX, ECX and DF are all set to 0

- g) In the next step, `lodsb` moves the first character from the string (in `ESI`) to `AL`. The first 'a' in the string is moved to `AL`.



AL now has the ascii value of the first character in the string 'a' (61) and the ESI register points to the address of the second character of the inputted string.

- h) The next step increments the value in AL by 11 and then compares the value in AL with 72. In this case the value of AL after incrementing by 11 is 72 so we know that the first character of the serial is indeed a lowercase 'a'. The jump to incorrect serial is not taken.



The EAX value is now 72 and we know the correct first character in the serial is ‘a’

- i) In the next step, the value in AX is moved to CX.

04_crackme.exe - PID: 1716 - Module: 04_crackme.exe - Thread: Main Thread 3968 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Nov 5 2020 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

```

mov esi,04_crackme.4027A5 esi:"aaaaaaa", 4027A5:"aaaaaaa"
cld
xor eax,eax
xor ecx,ecx
lodsb
add al,11
cmp al,72
jne 04_crackme.4010F7 72:'r'
mov cx,ax
lodsb
mul cx
cmp ax,2C88
jne 04_crackme.4010F7
lodsb
xor al,F0
cmp al,C3
jne 04_crackme.4010F7
lodsb
mov ecx,1B0
shr ecx,2
cmp al,C1
jne 04_crackme.4010F7
lodsb
cmp al,2D 2D:'-'
jne 04_crackme.4010F7
lodsd
xor eax,66666666
cmp eax,702550A
je 04_crackme.4010FE
mov eax,0

```

.text:004010C7 04_crackme.exe:\$10C7 #4C7

Hide FPU

EAX	00000072	'r'
EBX	00000001	
ECX	00000072	'r'
EDX	00000000	
EBP	000CFB18	&"AÜ\f"
ESP	000CFB08	
ESI	004027A6	"aaaaaaa"
EDI	00000000	
EIP	004010C7	04_crackme.004C

EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B

Default (stdcall) 5 Unlocked

1: [esp+4] 00000000
2: [esp+8] 00401022 04_crackme.004C
3: [esp+C] 00000001
4: [esp+10] 000CFB44 "AÜ\f"

The value of CX is now the same as AX, it's 72

- j) lodsb moves the next character from the address at ESI to AL, that is the second 'a' is moved to AL, AL becomes 61 again. The ESI register now holds the location of the next character in the string.

04_crackme.exe - PID: 1716 - Module: 04_crackme.exe - Thread: Main Thread 3968 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Nov 5 2020 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

```

mov esi,04_crackme.4027A5 esi:"aaaaaaa", 4027A5:"aaaaaaa"
cld
xor eax,eax
xor ecx,ecx
lodsb
add al,11
cmp al,72
jne 04_crackme.4010F7 72:'r'
mov cx,ax
lodsb
mul cx
cmp ax,2C88
jne 04_crackme.4010F7
lodsb
xor al,F0
cmp al,C3
jne 04_crackme.4010F7
lodsb
mov ecx,1B0
shr ecx,2
cmp al,C1
jne 04_crackme.4010F7
lodsb
cmp al,2D 2D:'-'
jne 04_crackme.4010F7
lodsd
xor eax,66666666
cmp eax,702550A
je 04_crackme.4010FE
mov eax,0

```

.text:004010C8 04_crackme.exe:\$10C8 #4C8

Hide FPU

EAX	00000061	'a'
EBX	00000001	
ECX	00000072	'r'
EDX	00000000	
EBP	000CFB18	&"AÜ\f"
ESP	000CFB08	
ESI	004027A7	"aaaaaaa"
EDI	00000000	
EIP	004010C8	04_crackme.004C

EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B

Default (stdcall) 5 Unlocked

1: [esp+4] 00000000
2: [esp+8] 00401022 04_crackme.004C
3: [esp+C] 00000001
4: [esp+10] 000CFB44 "AÜ\f"

AL now holds the value 61, the ascii of 'a' and ESI points to the third 'a' in the input string

- k) `mul CX` performs the operation $AX = AX * CX$. i.e, $AX = 61 * 72 = 2B32$. The next operation compares `AX` with `2C88`; therefore, the second character of the serial isn't 'a'. By working in reverse, we can identify the second character, $2C88 / 72 = 64$. '64' in ascii is 'd'.

04_crackme.exe - PID: 1716 - Module: 04_crackme.exe - Thread: Main Thread 3968 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Nov 5 2020 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

```

mov esi, 04_crackme.4027A5 esi: "aaaaaaa", 4027A5: "aaaaaaa"
cld
xor eax, eax
xor ecx, ecx
lodsb
add al, 11
cmp al, 72 72: 'r'
jne 04_crackme.4010F7
mov cx, ax
lodsb
mul cx
cmp ax, 2C88
jne 04_crackme.4010F7
lodsb
xor al, F0
cmp al, C3
jne 04_crackme.4010F7
lodsb
mov ecx, 1B0
shr ecx, 2
cmp al, cl
jne 04_crackme.4010F7
lodsb
cmp al, 2D 2D: '-'
jne 04_crackme.4010F7
lodsd
xor eax, 66666666
cmp eax, 702550A
je 04_crackme.4010FE
mov eax, 0

```

ax=2B32

.text:004010CB 04_crackme.exe:\$10CB #4CB

Hide FPU

EAX	00002B32
EBX	00000001
ECX	00000072 'r'
EDX	00000000
EBP	000CFB18 &"A0\rf"
ESP	000CFB08
ESI	004027A7 "aaaaaaa"
EDI	00000000
EIP	004010CB 04_crackme.004C

EFLAGS 00000202
ZF 0 PF 0 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_EXISTS)

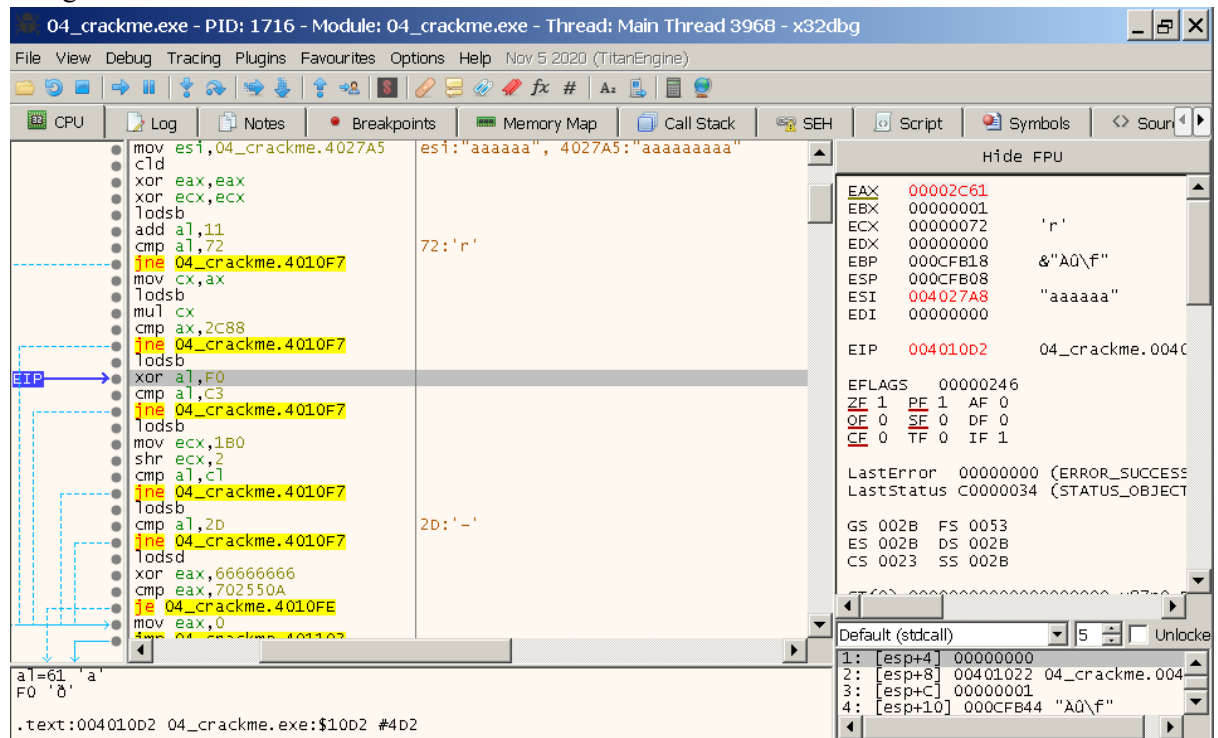
GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B

Default (stdcall) 5 Unlocked

1:	[esp+4]	00000000
2:	[esp+8]	00401022 04_crackme.004
3:	[esp+C]	00000001
4:	[esp+10]	000CFB44 "A0\rf"

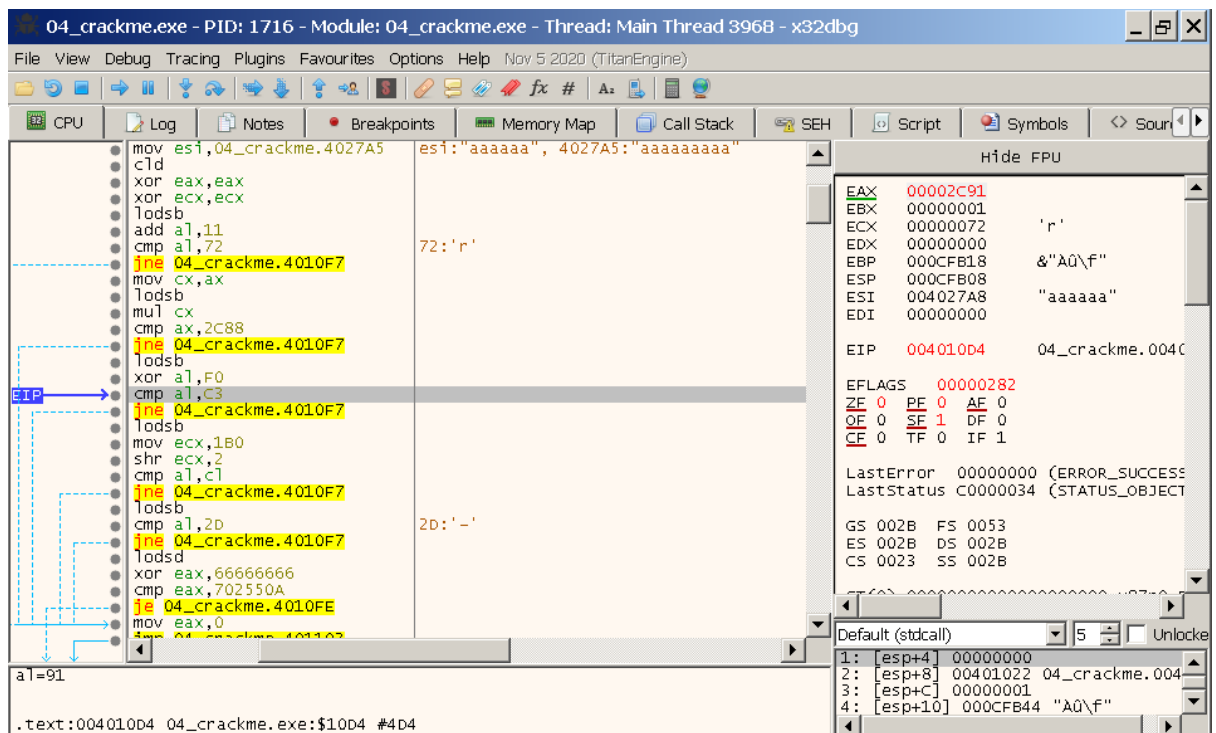
The value of `AX` after `mul CX` is `2B32`, but it should be `2C88`. By reverse engineering the operations we know that the second character in the serial is '64' or 'd'

- 1) By manually editing the EAX value to 2C88, the jump is not taken and the process continues. lodsb moves the character pointed at by ESI to AL, this makes EAX=2C61, 2C remains from the previous value in EAX and AL is updated to take the value of the next character in the string which is 'a'=61.



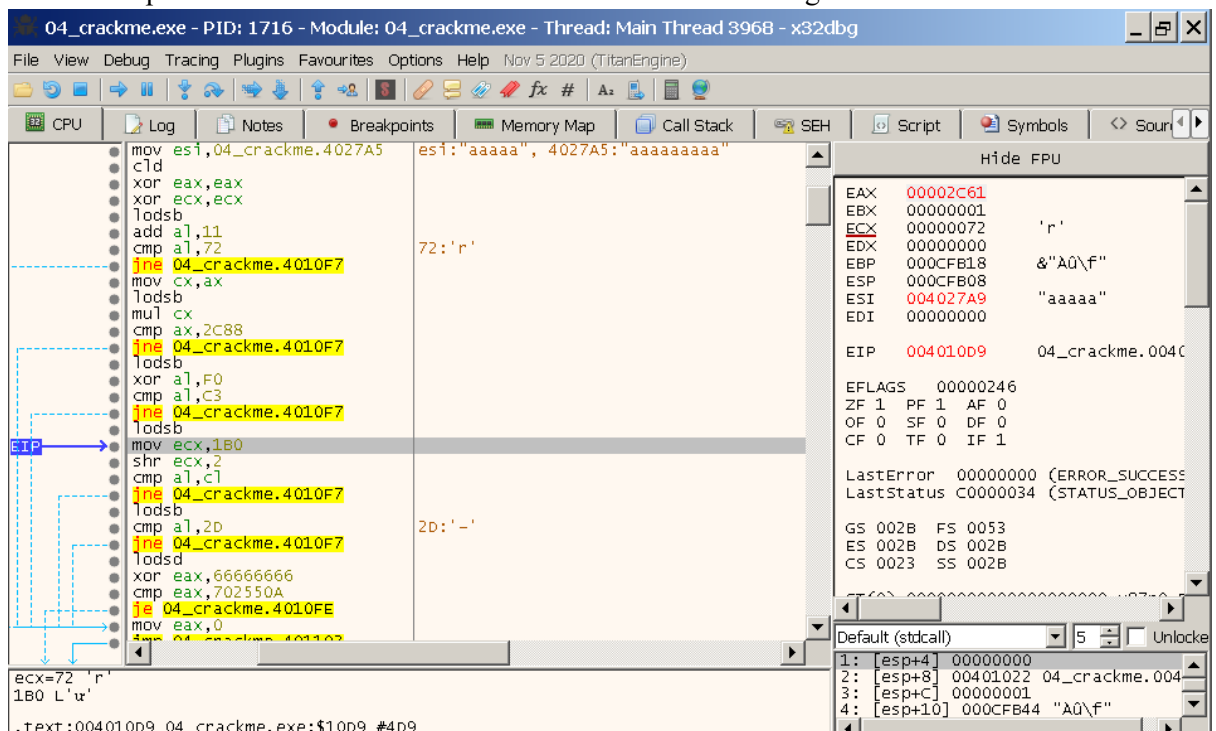
EAX is now 2C61

- m) The next action is to xor AL with F0 to obtain the value C3. Currently AL is 61 and $61 \text{ xor } F0 = 91$. Therefore, the third character in the serial is not 'a'. We can identify the third character by doing an xor operation on the result and F0. The third character is $F0 \text{ xor } C3 = 33$, which is '3'.



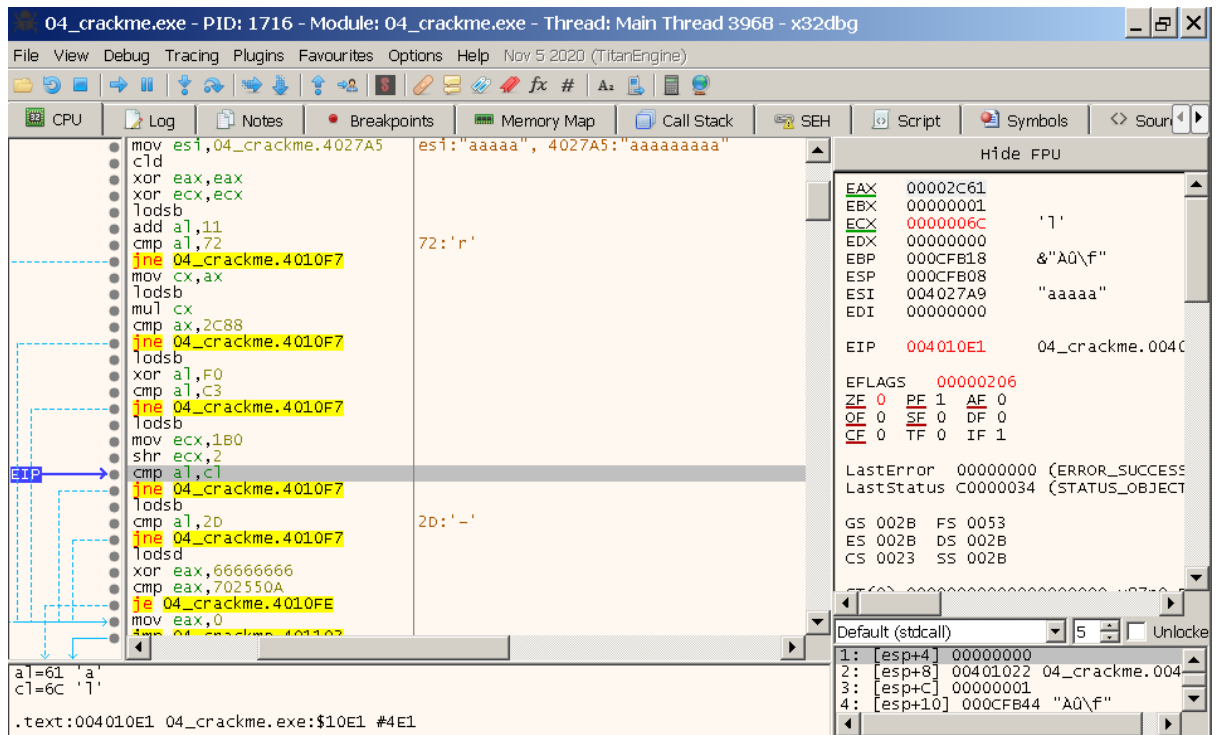
The value of AL after xor F0 is 91, but it should be C3. By reverse engineering the operations we know that the third character in the serial is '33' or '3'

- n) By manually editing the value of AL to C3, the jump is not taken. lodsb moves the character pointed at by ESI to AL, this makes EAX=2C61, 2C remains from the previous value in EAX and AL is updated to take the value of the next character in the string which is 'a'=61



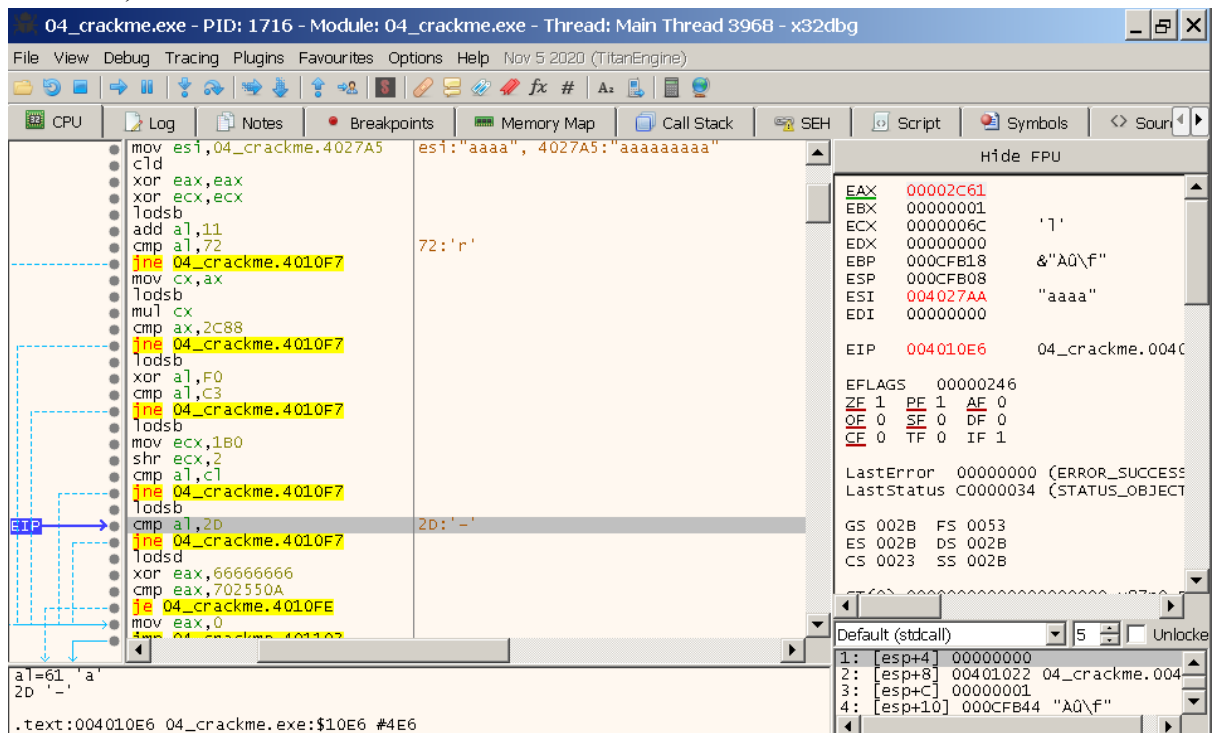
The value of EAX is 2C61

- o) Moving 1B0 to ECX and then shifting ECX right by 2 makes the value at ECX = 6C which is 'l'. Then the comparison is between AL and CL, as CL is 'l', the fourth character of the serial is 'l'.



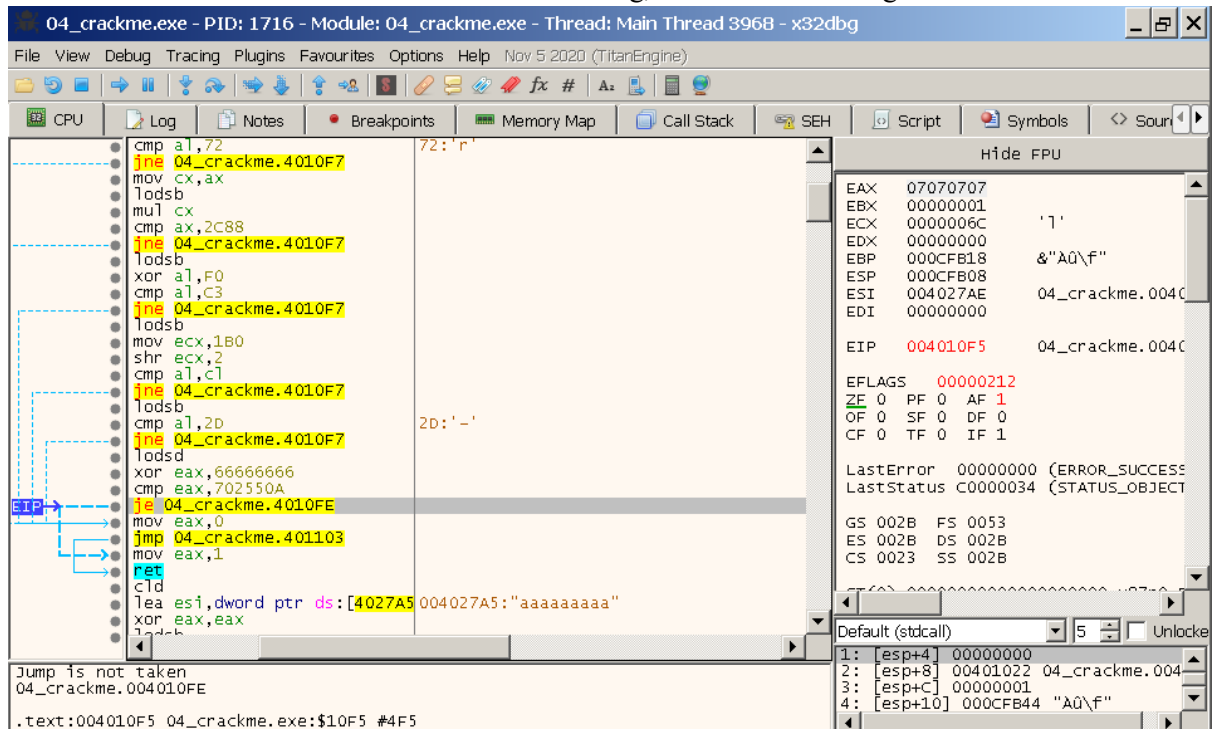
The value in ECX is 6C which is 'l'

- p) By manually editing the value of AL to 6C, the jump is not taken. lodsb moves the next character pointed at by ESI to AL. The next step directly compares AL to 2D which is '-'. Therefore, the fifth character in the serial is '-'.



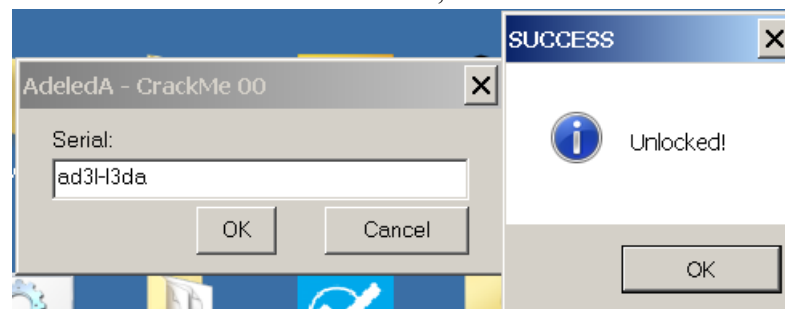
The value at AL is '61' for 'a' but the comparison shows that the correct value should be '2D' which is 'l'

- q) By manually editing the AL value to 2D, the jump is not taken. lodsd loads four bytes (all the remaining characters in the string) to EAX. As the direction flag is set to 0, it loads the string from the start of the memory block to the end, so if the remaining string was 'jklm' it would get loaded into EAX as 'mlkj'. The steps that follow are EAX xor 66666666 and then a comparison of EAX with 702550A. To obtain the correct value of EAX, we can xor 702550A with 66666666, the result of this operation is '6164336c' which is 'ad3l'. Because the string is loaded in reverse into EAX due to the direction flag, the actual remaining serial is 'l3da'.



If the final comparison is equal then the serial is validated

- r) Putting together all the identified serial characters, the serial is 'ad3l-l3da'.



Reference:

^[2]<https://x64dbg.com/#start>

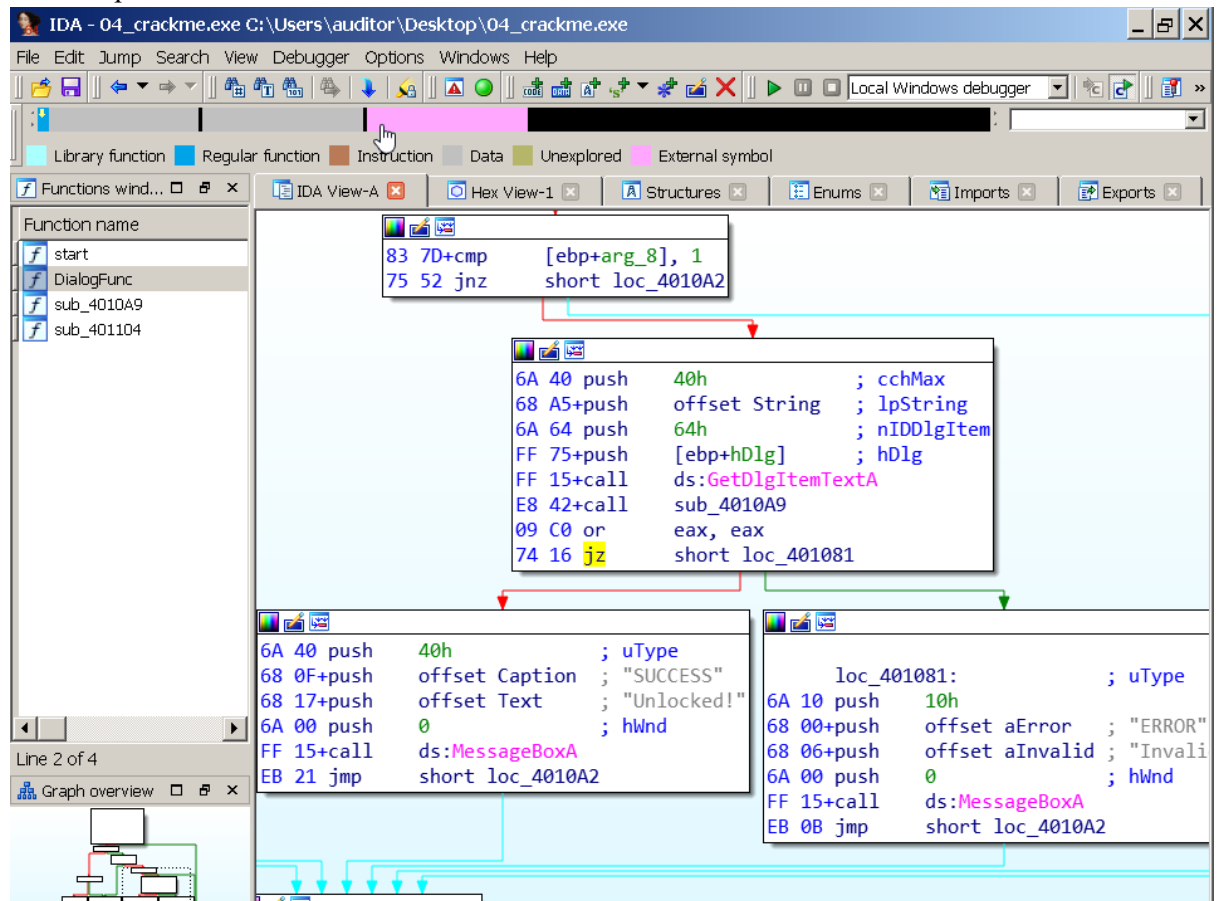
Patching the program

Description:

Using a debugger, we can edit the opcodes to change the flow of the program. This modified program can be saved to obtain a patched version which works without the correct serial.

Process:

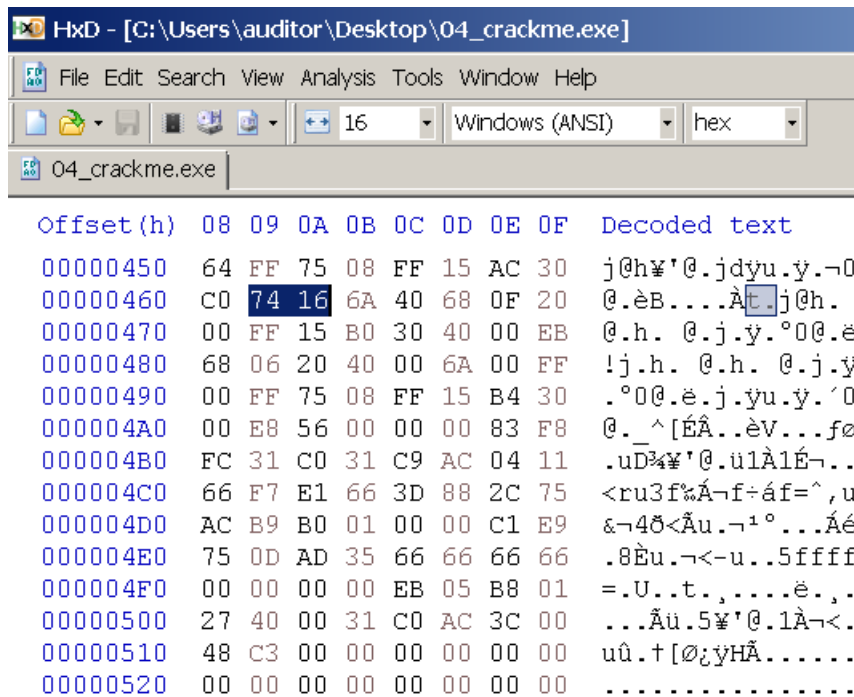
- a) Using IDA open the program and change the number of opcode bytes to 2 to see the opcodes. The JZ opcode here is 74 16.



Editing the opcode here changes the flow of the program

- b) Open the program in a hexeditor^[3] by mh-nexus and search for the opcode 74 16. Then change the hexcode to change where the program jumps. Referencing an opcode chart, 74 is jz and 75 is jnz. By editing the opcode to 75, the program will unlock for any entered serial except for the correct serial.

74 is JZ and 75 is JNZ



HxD - [C:\Users\auditor\Desktop\04_crackme.exe]

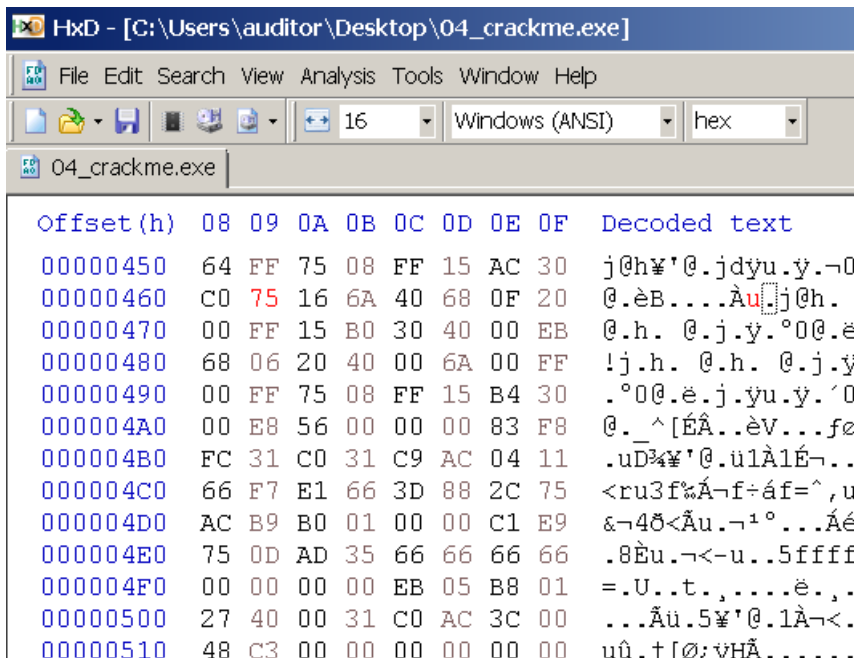
File Edit Search View Analysis Tools Window Help

16 Windows (ANSI) hex

04_crackme.exe

Offset(h)	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000450	64	FF	75	08	FF	15	AC	30	j@h%*@.jdyu.y.-0
00000460	C0	74	16	6A	40	68	0F	20	@.èB....À[j@h.
00000470	00	FF	15	B0	30	40	00	EB	@.h. @.j.y.°0@.è
00000480	68	06	20	40	00	6A	00	FF	!j.h. @.h. @.j.y
00000490	00	FF	75	08	FF	15	B4	30	.°0@.è.j.yu.y.°0
000004A0	00	E8	56	00	00	00	83	F8	@._^[ÉÂ...èV...fø
000004B0	FC	31	C0	31	C9	AC	04	11	.uD%*@.ùlÀlÉ¬..
000004C0	66	F7	E1	66	3D	88	2C	75	<ru3f%Á¬f÷áf=^,u
000004D0	AC	B9	B0	01	00	00	C1	E9	&¬4ð<Àu.¬¹°...Áé
000004E0	75	0D	AD	35	66	66	66	66	.8Èu.¬<-u..5ffff
000004F0	00	00	00	00	EB	05	B8	01	=.U..t.,....è.,.
00000500	27	40	00	31	C0	AC	3C	00	...Àù.5%*@.lÀ¬<.
00000510	48	C3	00	00	00	00	00	00	uû.+[ø;yHÃ.....
00000520	00	00	00	00	00	00	00	00

Finding the opcodes in the hexeditor



HxD - [C:\Users\auditor\Desktop\04_crackme.exe]

File Edit Search View Analysis Tools Window Help

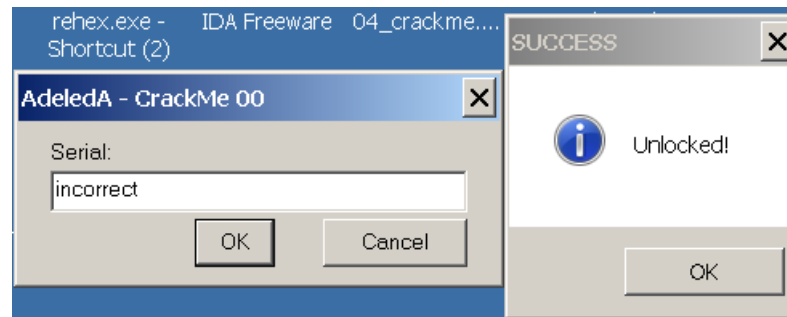
16 Windows (ANSI) hex

04_crackme.exe

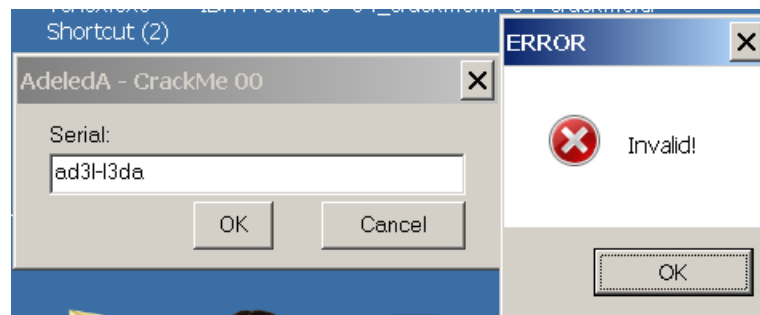
Offset(h)	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000450	64	FF	75	08	FF	15	AC	30	j@h%*@.jdyu.y.-0
00000460	C0	75	16	6A	40	68	0F	20	@.èB....Àu[j@h.
00000470	00	FF	15	B0	30	40	00	EB	@.h. @.j.y.°0@.è
00000480	68	06	20	40	00	6A	00	FF	!j.h. @.h. @.j.y
00000490	00	FF	75	08	FF	15	B4	30	.°0@.è.j.yu.y.°0
000004A0	00	E8	56	00	00	00	83	F8	@._^[ÉÂ...èV...fø
000004B0	FC	31	C0	31	C9	AC	04	11	.uD%*@.ùlÀlÉ¬..
000004C0	66	F7	E1	66	3D	88	2C	75	<ru3f%Á¬f÷áf=^,u
000004D0	AC	B9	B0	01	00	00	C1	E9	&¬4ð<Àu.¬¹°...Áé
000004E0	75	0D	AD	35	66	66	66	66	.8Èu.¬<-u..5ffff
000004F0	00	00	00	00	EB	05	B8	01	=.U..t.,....è.,.
00000500	27	40	00	31	C0	AC	3C	00	...Àù.5%*@.lÀ¬<.
00000510	48	C3	00	00	00	00	00	00	uû.+[ø;yHÃ.....

Editing the opcode to make the operation JNZ

- c) Save the edited program and execute the patched version to test whether it unlocks for any serial.



Random string unlocks the application



The correct serial no longer works to unlock the application

Reference:

^[3] <https://mh-nexus.de/en/>