

Task 1. Разобрать пример реверс-инжиниринга файла CrackMe.

Тут необходимо внимательно изучить процесс обратной разработки для файла crackme_example.exe. Пример этого разбора приложен в виде файла к третьему уроку. Повторить этот же процесс самостоятельно, используя программный комплекс IDA Pro.

В качестве ответа к этому заданию необходимо прислать валидный серийный номер, который проходит проверку файлом и который отличается от того, что показан в примере. Если внимательно изучить пример, то можно легко составить еще один номер, на основе предложенного.

Бонусное задание (сделайте по желанию): написать keygen – генератор случайных кодов, который будет проверять их на корректность, используя правила данного CrackMe, на удобном языке программирования. Прислать код этой программы и её вывод – валидные серийные ключи.

```
; +-----+
; | This file was generated by The Interactive Disassembler (IDA) |
; | Copyright (c) 2023 Hex-Rays, <support@hex-rays.com> |
; | Freeware version |
; +-----+
;
; Input SHA256 : 782F1A79B1F40AB7807B8F28180680ACE6774703C62E6A83500FEA3D595BB3CB
; Input MD5 : 3ECE7EBDCE58A824D5E07496E8C847E0
; Input CRC32 : 7C1D50B9

; File Name : C:\Users\Admin\Downloads\crackme_example (1).exe
; Format : Portable executable for AMD64 (PE)
; Imagebase : 140000000
; Timestamp : 49DCD58C (Wed Apr 08 16:49:16 2009)
; Section 1. (virtual address 00001000)
; Virtual size : 00000271 ( 625.)
; Section size in file : 00000400 ( 1024.)
; Offset to raw data for section: 00000400
; Flags 60000020: Text Executable Readable
; Alignment : default

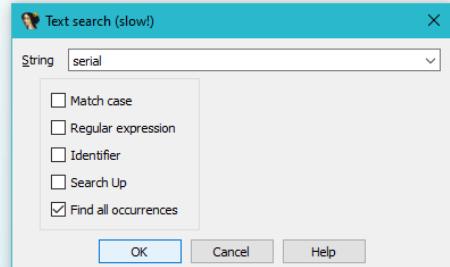
.686p
.mmx
.model flat

; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 14000100h
assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing

sub_14000100 proc near

var_18= byte ptr -18h
var_8= qword ptr -8

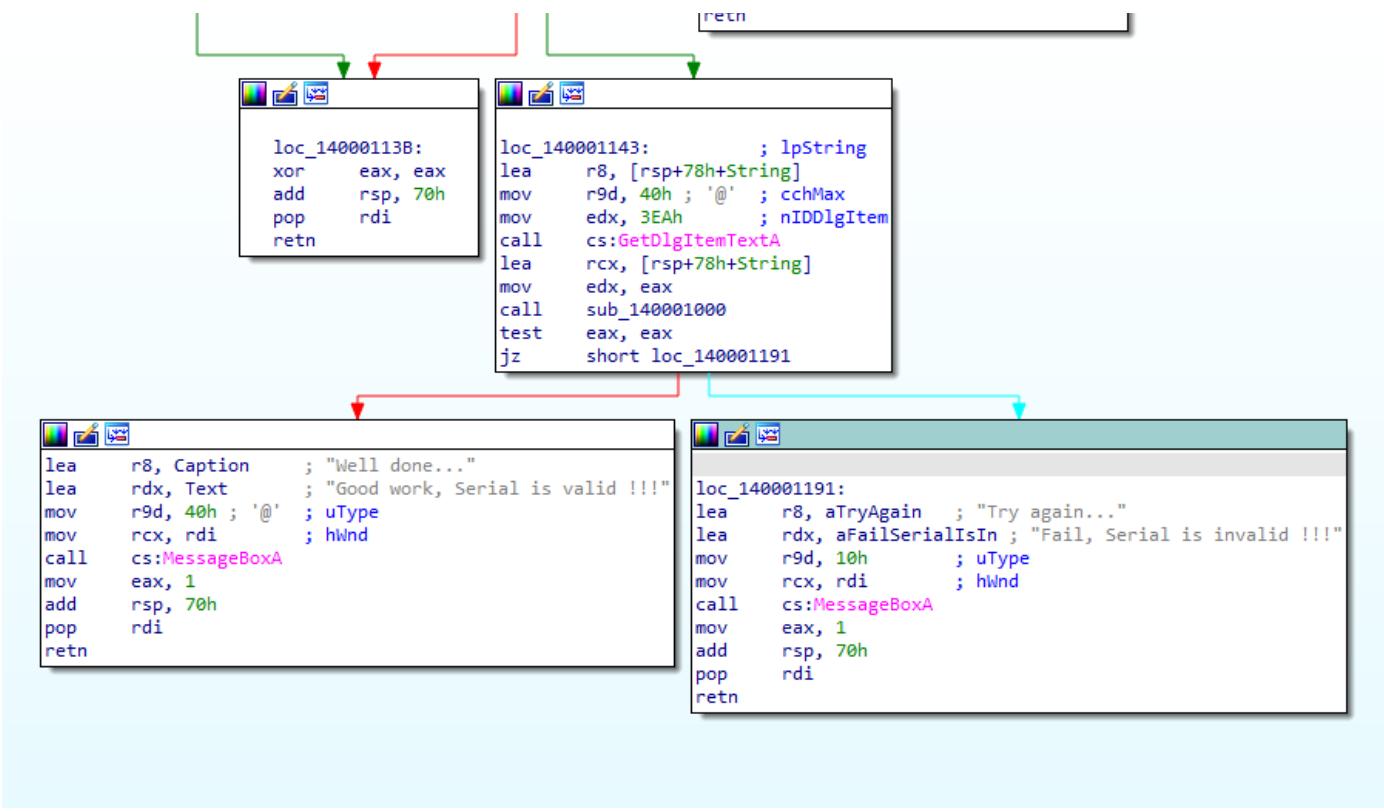
sub    rsp, 18h
cmp    edx, 13h
mov    r8, rcx
jz     short loc_140001013
```



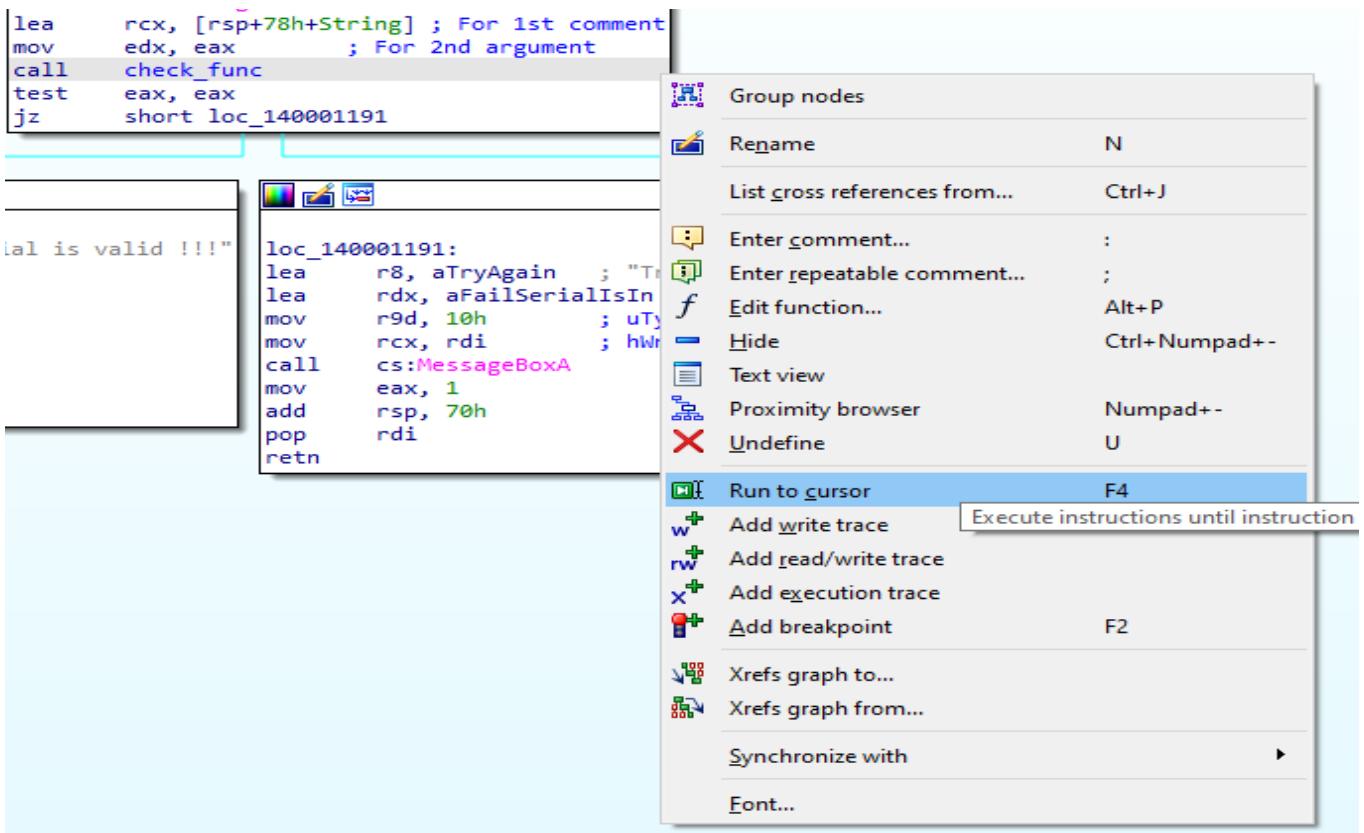
Тут я нажал «alt+t» и набрал в поиске serial + с опцией «Найти все проишествия».

Address	Function	Instruction
.text:0000000140001170	DialogFunc	lea rdx, Text ; "Good work, Serial is valid !!!"
.text:0000000140001198	DialogFunc	lea rdx, aFailSerialIsIn ; "Fail, Serial is invalid !!!"
.rdata:0000000140002070		Text db 'Good work, Serial is valid !!!',0
.rdata:00000001400020A0		; const CHAR aFailSerialIsIn[]

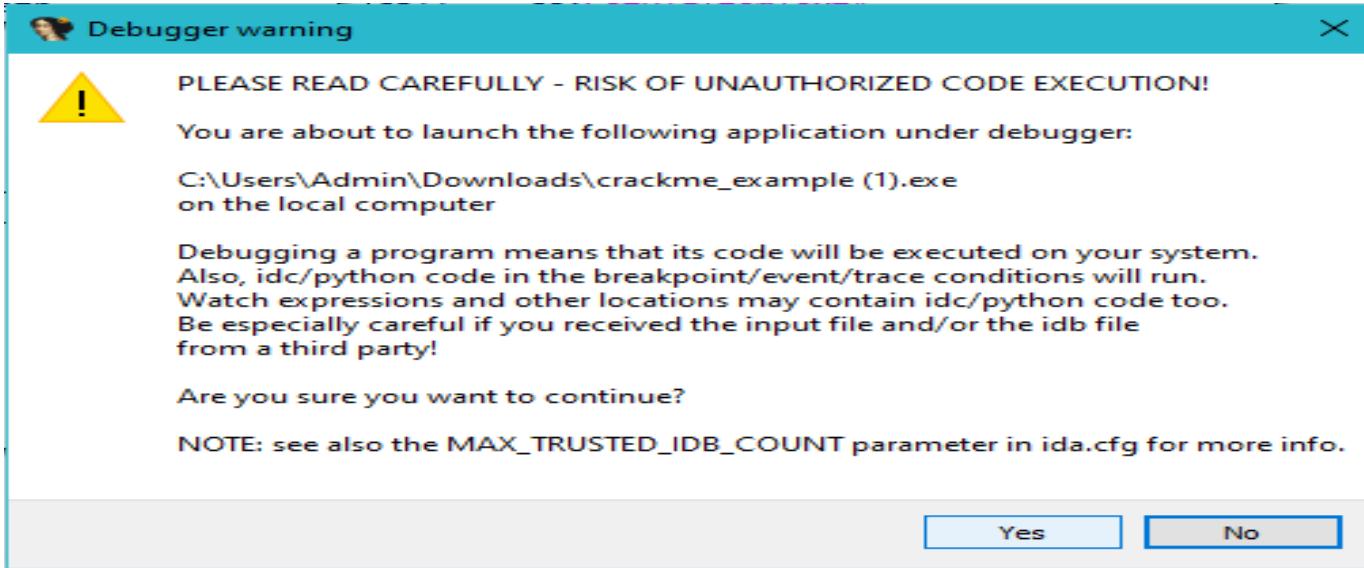
Появились данные запросы, тут интересен запрос на недействительный серийный — поэтому нажимаю на него.



Вылезло данное меню, но тут нужно окно с функцией, вызывающий pop-up с недействительным серийным.



Выделить функцию, которую я переименовал в «check_func» и нажал на «Запустить на курсоре» (переименовывается выделением функции + кнопка N + переименовывание самой функции).



В первый раз появляется данное предупреждение об риске несанкционированного исполнения кода.

0000000000014EC70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000014EC80	00 03 0D 00 00 00 00 00 59 11 00 40 01 00 00 00Y...@....
0000000000014EC90	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000014ECA0	00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00@.....
0000000000014ECB0	31 32 33 34 35 00 00 00 00 00 00 00 00 00 00 00 12345.....
0000000000014ECC0	00 00 00 00 70 3E 00 00 00 00 00 00 00 00 00 00p>.....
0000000000014ECD0	01 00 00 00 00 00 00 00 AE 2A 33 19 F8 7F 00 00*3.....
0000000000014ECE0	F8 15 57 00 00 00 00 00 00 00 00 00 00 00 00 00 ..W.....
0000000000014ECF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

В случае с HEX, показываются введённые данные на crackme-example.exe

General registers	
RAX 0000000000000005	^
RBX 0000000000000001	^
RCX 0000000000014ECB0	↳ Stack[00004EF8]:0000000000014ECB0
RDX 0000000000000005	^
RSI 0000000000000000	^
RDI 00000000000D0300	↳ debug003:00000000000D0300
RBP 0000000000014EE68	↳ Stack[00004EF8]:0000000000014EE68
RSP 0000000000014EC90	↳ Stack[00004EF8]:0000000000014EC90
RIP 0000000140001160	↳ DialogFunc+50
R8 0000000000014EAF0	↳ Stack[00004EF8]:0000000000014EAF0
R9 0000000000000001	^
R10 000000000016D1190	↳ debug069:000000000016D1190

В случае с просыами регистрами, после 14ECB0, показываются значение 000005.

Serial

12345678

Check

А набрав другое значение, например такое.

0000000000014EC70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000014EC80	B6 06 0A 00 00 00 00 00 59 11 00 40 01 00 00 00Y..@....
0000000000014EC90	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000014ECA0	00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00@.....
0000000000014ECB0	31 32 33 34 35 36 37 38 00 00 00 00 00 00 00 00	12345678.....
0000000000014ECC0	00 00 00 00 CC 04 00 00 00 00 00 00 00 00 00 00 00
0000000000014ECD0	01 00 00 00 00 00 00 00 AE 2A 33 19 F8 7F 00 00*3.....
0000000000014ECE0	F8 15 45 00 00 00 00 00 00 00 00 00 00 00 00 00	.E.....
0000000000014ECF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

В случае с HEX, введённые данные тоже поменялись.

General registers

RAX 00000000000000000000000000000008	↳
RBX 00000000000000000000000000000001	↳
RCX 0000000000014ECB0	↳ Stack[00003E18]:0000000000014ECB0
RDX 00000000000000000000000000000008	↳
RSI 00000000000000000000000000000000	↳
RDI 00000000000A06B6	↳ debug003:000000000000A06B6
RBP 0000000000014EE68	↳ Stack[00003E18]:0000000000014EE68
RSP 0000000000014EC90	↳ Stack[00003E18]:0000000000014EC90
RIP 0000000140001160	↳ DialogFunc+50
R8 0000000000014EAFO	↳ Stack[00003E18]:0000000000014EAFO
R9 0000000000000001	↳
R10 00000000017D7F00	↳ debug072:0000000000017D7F00

Тут тоже самое.

```
.text:0000000140001013 ; -.-.
.text:0000000140001013
.text:0000000140001013 loc_140001013:
        xor    edx, edx          ; CODE XREF: check_func+A1j
.text:0000000140001013
.text:0000000140001015     lea    rax, [r8+4]
.text:0000000140001019     mov    ecx, edx
.text:0000000140001018     xchg   ax, ax
.text:000000014000101D     db    66h, 66h
.text:000000014000101D     xchg   ax, ax
.text:0000000140001020
.text:0000000140001020 loc_140001020:
        cmp    byte ptr [rax], 20h ; '-'
        short loc_14000100C
.text:0000000140001023     jnz    ecx, 1
.text:0000000140001025     add    rax, 5
.text:0000000140001028     add    ecx, 3
.text:000000014000102C     cmp    ebx, [rsp+18h+var_8]
        jb    short loc_140001020
.text:0000000140001031
.text:0000000140001031 loc_140001031:
        mov    [rsp+18h+var_8], rbx
        ; DATA XREF: .rdata:00000001400020C4↓o
        ; .rdata:00000001400020D8↓o ...
.text:0000000140001031
.text:0000000140001031     mov    r10d, edx
.text:0000000140001036     mov    r11d, edx
.text:0000000140001039     mov    rbx, [rsp+18h+var_18]
.text:000000014000103C     lea    rsi, r8
.text:0000000140001040     mov    r9, r8
.text:0000000140001043     nop
.text:0000000140001044     db    66h, 66h
.text:0000000140001044     xchg   ax, ax
.text:0000000140001048     db    66h, 66h
.text:0000000140001048     xchg   ax, ax
.text:000000014000104C     db    66h, 66h
.text:000000014000104C     xchg   ax, ax
.text:0000000140001050
```

Важные строки, которые показывают алгоритм.

```

.text:00000014000104C          db   66h, 66h
.text:00000014000104C          xchg  ax, ax
.text:000000140001050         ; CODE XREF: check_func+A2+j
.loc_140001050:             mov   rcx, rdx
.text:000000140001053         ; CODE XREF: digital_check+6C+j
.loc_140001053:              movsx eax, byte ptr [r9+rcx]
.add  eax, 0FFFFFD0h
 cmp   eax, 9
 ja    loc_1400010FD
 add   rcx, 1
 cmp   rcx, 4
 jl    short _digital_check
 movsx eax, byte ptr [r9]
 add   al, [r9+1]
 add   al, [r9+2]
 movsx ecx, byte ptr [r9+3]
 add   eax, ecx
 add   eax, ecx
 add   eax, ecx
 add   rbx, 4
 add   r10d, 1
 add   r9, 5
 lea   ecx, [rcx+rax-150h]
 mov   [rbx-4], ecx
 add   r10d, ecx
 cmp   r10d, 4
 jb   short loc_140001050
 shr   r10d, 2
 mov   ecx, edx
 lea   rax, [rsp+18h+var_18]
 xchg ax, ax

```

```

.text:0000001400010A8          mov   ecx, edx
.text:0000001400010AA          lea   rax, [rsp+18h+var_18]
.text:0000001400010AE          xchg ax, ax
.loc_1400010B0:              ; CODE XREF: check_func+BF+j
 cmp   [rax], r10d
 jnz  short loc_1400010FD
 add   ecx, 1
 add   rax, 4
 cmp   ecx, 4
 jb   short loc_1400010B0
 mov   rax, rdx
.text:0000001400010C4         ; CODE XREF: equal_check+EC+j
.text:0000001400010C4         movzx eax, byte ptr [r8+rax+5]
.text:0000001400010CA          cmp   [r8+rax], cl
.jz   short loc_1400010FD
.text:0000001400010CE          movzx r9d, byte ptr [rax+r8+0Ah]
 cmp   cl, r9b
.jz   short loc_1400010FD
.text:0000001400010D6          cmp   r9b, [rax+r8+0Fh]
.jz   short loc_1400010FD
.text:0000001400010D9          add   edx, 1
.rax 1
.text:0000001400010D8          add   rax, 1
 cmp   edx, 4
.jb   short equal_check
.text:0000001400010E0          mov   eax, 1
.text:0000001400010E2          mov   rbx, [rsp+18h+var_8]
.add  rsp, 18h
.retn
.text:0000001400010E5          ; -----
.text:0000001400010E9          ; -----
.text:0000001400010EC          ; -----
.text:0000001400010EE          ; -----
.text:0000001400010F3          ; -----
.text:0000001400010F8          ; -----
.text:0000001400010FC          ; -----
.text:0000001400010FD          ; -----
.loc_1400010FD:              ; CODE XREF: check_func+5E+j

```

Общий расчёт: считывание трёх разных цифр от 0 до 9 по одному разу и четвёртую цифру в общем количестве тоже четыре раза, затем в конце отнять на 150h (по шестнадцатеричной системе считывания). Состоит из 19 знаков ASCII: XXXX-XXXX-XXXX-XXXX.

Примерно так: 31h+32h+33h+34h+34h+34h+34h-150h=16h

При получении действительного серийного, важно чтобы все комбинации были равны одному и тому же шестнадцатеричному числу 4 раза подряд.

Последним осталось, это написать действительный серийный номер, но некоторое время, я не мог разобраться. После некоторого времени, я смог разобраться — как нужно считывать.

калькуляторов Арифметика двоичных чисел и Перевод дробных чисел из одной системы счисления в другую и сделать универсальный калькулятор, который может выполнять основные математические действия (сложение, вычитание, умножение, деление и возведение в степень) над числами в любой системе счисления. Для указания системы счисления используется параметр «Основание системы счисления», в которой записано выражение, в котором можно указать любое число от 2 до 36. Например, 2 для двоичной, 8 для восьмеричной, 16 для шестнадцатеричной и так далее.

Также поддерживаются выражения с дробными числами. Поскольку все вычисления реализованы через десятичную систему счисления, результаты для дробных чисел не всегда могут быть точны. Точность преобразования можно задавать параметром «Точность преобразования дробных чисел (разрядов)». Прочитать по поводу точности преобразования можно здесь Перевод дробных чисел из одной системы счисления в другую

Для возвведения в степень используется конструкция вида число^{степень} (внизу на примере — 110¹⁰).

Калькулятор с поддержкой разных систем счисления

Выражение для расчета
30+37+37+34+34+34+34-150

Система счисления
16

Точность вычисления
Знаков после запятой: 0

Основание системы счисления, в которой записано выражение

РАССЧИТАТЬ

Результат вычисления (в указанной системе счисления (десятичный))
1E **30**

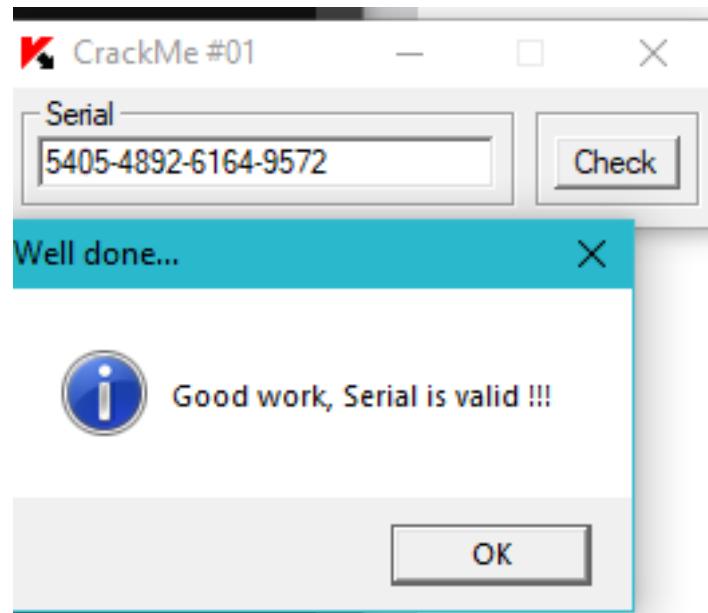
Я пользовался данным калькулятором. (На данном экранном выстреле, я решил набрать

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		Del

asciicharstable.com

серийный, у которого 4 комбинации будут равны 1E по шестнадцатеричной системе.)

И данной ASCII/HEX таблицей.



Перебрав некоторые комбинации, следуя данному алгоритму — я смог набрать действительный серийный.

Task 2. Анализ другого CrackMe-файла.

Провести самостоятельный анализ еще одного файла crackme_homework.exe по тому же алгоритму, что и разобранный пример.

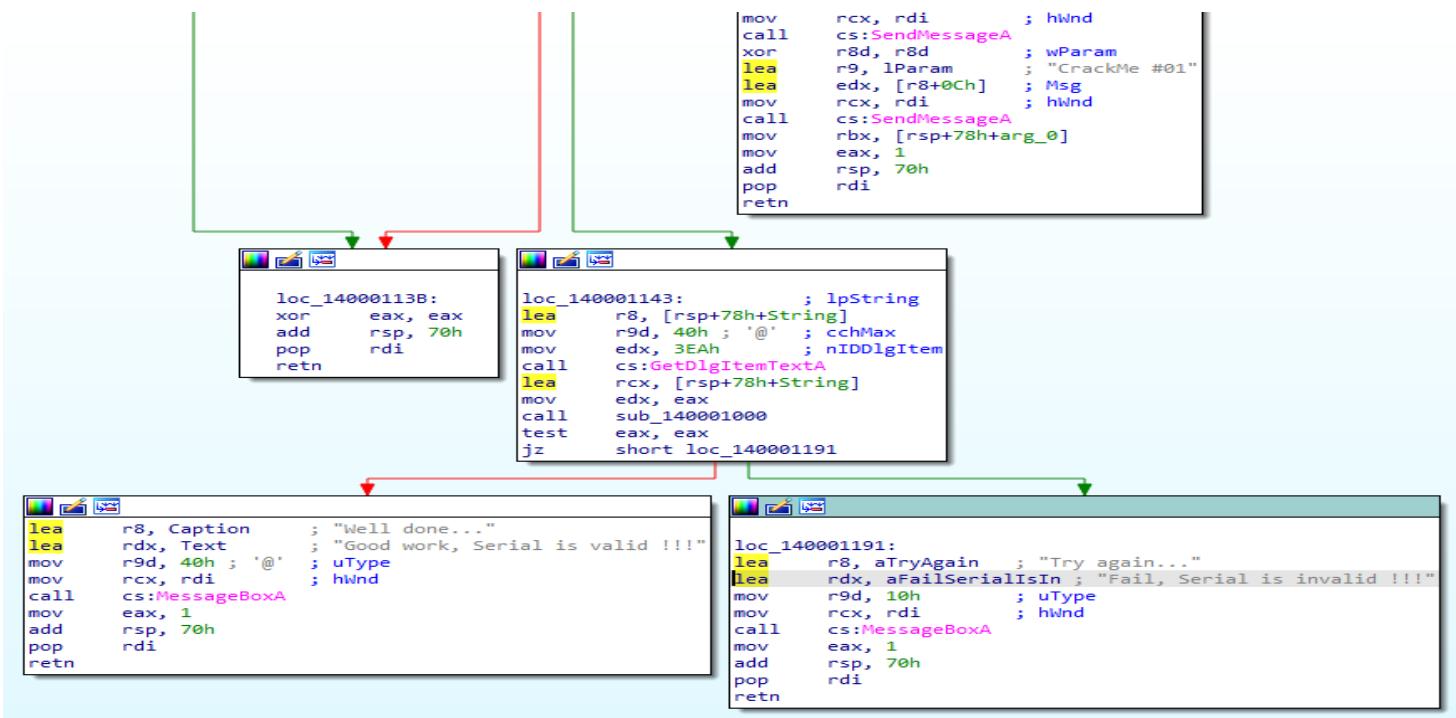
В качестве ответа приложить отчёт о проведении процесса обратной разработки.

Отчёт должен содержать:

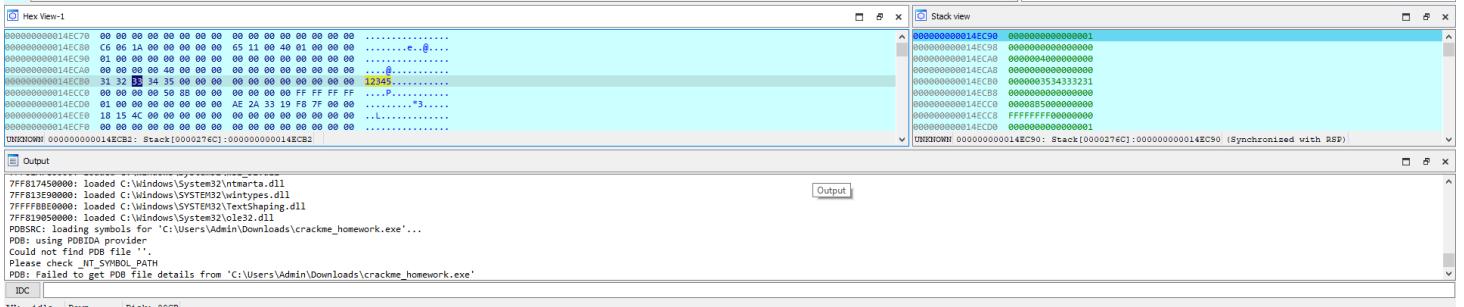
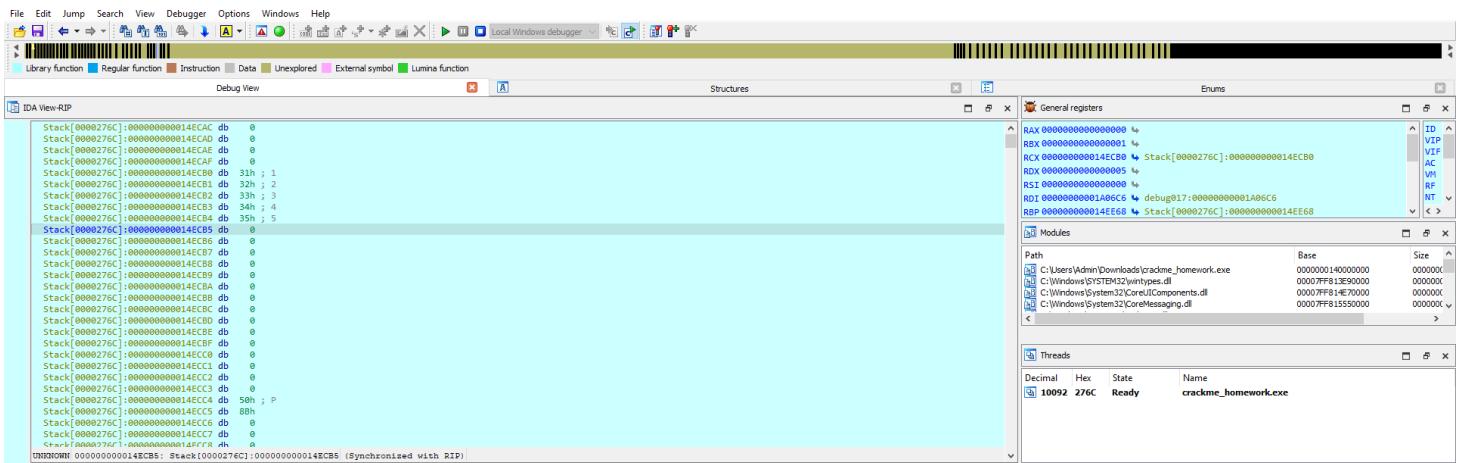
- формальное описание функциональности разбираемого файла (что делает программа с указанием функций);
- формат вводимого серийного номера для проверки (например, строка из 19 ASCII символов формата XXXX-XXXX-XXXX-XXXX, где X — прописная буква латинского алфавита);
- как можно больше найденных условий, позволяющих генерировать серийный номер, проходящий проверку;
- (не обязательно) пример валидного серийного номера.

Подкрепите свой отчёт скриншотами дизассемблера, где будут содержаться важные для анализа строки (желательно с комментариями к ним).

Address	Function	Instruction
.text:0000000140001170	DialogFunc	lea rdx, Text ; "Good work, Serial is valid !!!"
.text:0000000140001198	DialogFunc	lea rdx, aFailSerialIsIn ; "Fail, Serial is invalid !!!"
.rdata:0000000140002070		Text db 'Good work, Serial is valid !!!',0
.rdata:00000001400020A0		; const CHAR aFailSerialIsIn[]



Тут я также нажал на поиск, serial и в поиске на недействительный серийный.
Содержимое данного файла (crackme_homework.exe) выглядит так.



Тут я также набрал 12345 в обработчике (или дебагgere) — тут я показал полный интерфейс. В hex и обычных регистрах, указываются набранные данные.

```

.text:0000000140001141 pop    rdi
.text:0000000140001142 retn
.text:0000000140001143 ;
.text:0000000140001143 loc_140001143:           ; CODE XREF: DialogFunc+29tj
.text:0000000140001143 lea     r8, [rsp+78h+String]   ; lpString
.text:0000000140001148 mov    r9d, 40h ; '@'          ; cchMax
.text:000000014000114E mov    edx, 3EAh           ; nIDDigItem
.text:0000000140001153 call   cs:GetDlgItemTextA
.text:0000000140001159 lea     rcx, [rsp+78h+String]
.text:000000014000115E mov    edx, eax
.text:0000000140001160 call   sub_140001000
.text:0000000140001165 test   eax, eax
.text:0000000140001167 jz    short check_func
.text:0000000140001169 lea     r8, Caption          ; "Well done..."
.text:0000000140001170 lea     rdx, Text            ; "Good work, Serial is valid !!!"
.text:0000000140001177 mov    r9d, 40h ; '@'          ; uType
.text:000000014000117D mov    rcx, rdi           ; hWnd
.text:0000000140001180 call   cs:MessageBoxA
.text:0000000140001186 mov    eax, 1
.text:0000000140001188 add    rsp, 70h
.text:000000014000118F pop    rdi
.text:0000000140001190 retn
.text:0000000140001191 ;
.text:0000000140001191 check_func:                 ; CODE XREF: DialogFunc+57tj
.text:0000000140001191 lea     r8, aTryAgain       ; "Try again..."
.text:0000000140001198 lea     rdx, aFailSerialIsIn ; "Fail, Serial is invalid !!!"
.text:000000014000119F mov    r9d, 10h           ; uType

```

Алгоритм данного файла проверяет на латинские буквы. Буквы в шестнадцатеричной системе: от 41 до 5A и от 61 до 7A.

```
.text:0000000140001007 mov    r8, rcx
.text:000000014000100A jz     short loc_140001013
.text:000000014000100C
.text:000000014000100C loc_14000100C; CODE XREF: sub_140001000+23↓j
.text:000000014000100C xor    eax, eax
.text:000000014000100E add    rsp, 18h
.text:0000000140001012 retn
.text:0000000140001013 ; -----
.text:0000000140001013 loc_140001013; CODE XREF: sub_140001000+A↓j
.text:0000000140001013 xor    edx, edx
.text:0000000140001015 lea    rax, [r8+4]
.text:0000000140001019 mov    ecx, edx
.text:000000014000101B xchg   ax, ax
.text:000000014000101D db    66h, 66h
.text:000000014000101D xchg   ax, ax
.text:0000000140001020
.text:0000000140001020 loc_140001020; CODE XREF: sub_140001000+2F↓j
.text:0000000140001020 cmp    byte ptr [rax], 20h ; '-'
.text:0000000140001023 jnz    short loc_14000100C
.text:0000000140001025 add    ecx, 1
.text:0000000140001028 add    rax, 5
.text:000000014000102C cmp    ecx, 3
.text:000000014000102F jb    short loc_140001020
.text:0000000140001031
.text:0000000140001031 loc_140001031; DATA XREF: .rdata:00000001400020C4↓o
.text:0000000140001031
.text:0000000140001031 mov    [rsp+18h+var_8], rbx
+rvt.0000000140001036 mov    r10d, adv
```

По количеству символов, тут такое же количество, что и в прошлом файле.

```
.text:000000014000106C jl    short digital_check
.text:000000014000106E movsx  eax, byte ptr [r9+2]
.text:0000000140001073 movsx  ecx, byte ptr [r9+3]
.text:0000000140001078 add    r11d, 1
.text:000000014000107C add    ecx, eax
.text:000000014000107E movsx  eax, byte ptr [r9+1]
.text:0000000140001083 add    rbx, 4
.text:0000000140001087 add    ecx, eax
.text:0000000140001089 movsx  eax, byte ptr [r9]
.text:000000014000108D add    r9, 5
.text:0000000140001091 lea    ecx, [rcx+rax-0C0h]
.text:0000000140001098 mov    [rbx-4], ecx
.text:000000014000109B add    r10d, ecx
.text:000000014000109E cmp    r11d, 4
.text:00000001400010A2 jb    short loc_140001050
.text:00000001400010A4 shr    r10d, 2
.text:00000001400010A8 mov    ecx, edx
.text:00000001400010AA lea    rax, [rsp+18h+var_18]
.text:00000001400010AE xchg   ax, ax
.text:00000001400010B0
.text:00000001400010B0 loc_1400010B0; CODE XREF: sub_140001000+BF↓j
.text:00000001400010B0 cmp    [rax], r10d
.text:00000001400010B3 jnz    short loc_1400010FD
.text:00000001400010B5 add    ecx, 1
.text:00000001400010B8 add    rax, 4
.text:00000001400010BC cmp    ecx, 4
.text:00000001400010BF jb    short loc_1400010B0
.text:00000001400010C1 mov    rax, rdx
+rvt.00000001400010C4
```

Другие важные строки

```

.text:00000001400010C1 mov rax, rdx
.text:00000001400010C4 loc_1400010C4: ; CODE XREF: sub_140001000+EC↓j
.text:00000001400010C4 movzx ecx, byte ptr [r8+rax+5]
.text:00000001400010CA cmp [r8+rax], cl
.text:00000001400010CE jz short loc_1400010FD
.text:00000001400010D0 movzx r9d, byte ptr [rax+r8+0Ah]
.text:00000001400010D6 cmp cl, r9b
.text:00000001400010D9 jz short loc_1400010FD
.text:00000001400010D8 cmp r9b, [rax+r8+0Fh]
.text:00000001400010E0 jz short loc_1400010FD
.text:00000001400010E2 add edx, 1
.text:00000001400010E5 add rax, 1
.text:00000001400010E9 cmp edx, 4
.text:00000001400010EC jb short loc_1400010C4
.text:00000001400010EE mov eax, 1
.text:00000001400010F3 mov rbx, [rsp+18h+var_8]
.text:00000001400010F8 add rsp, 18h
.text:00000001400010FC retn
.text:00000001400010FD ;
.text:00000001400010FD loc_1400010FD: ; CODE XREF: sub_140001000+5E↑j
.text:00000001400010FD ; sub_140001000+B3↑j ...
.text:00000001400010FD xor eax, eax
.text:00000001400010FF mov rbx, [rsp+18h+var_8]
.text:0000000140001104 add rsp, 18h
.text:0000000140001108 retn
.text:0000000140001108 sub_140001000 endp
+evt::aaaaaaaa140001108

```

Это является условием для составления действительного серийного: при получении действительного серийного, важно чтобы все комбинации были равны одному и тому же шестнадцатеричному числу 4 раза подряд.

```

.text:0000000140001110 ; ===== S U B R O U T I N E =====
.text:0000000140001110
.text:0000000140001110
.text:0000000140001110 ; INT PTR _stdcall DialogFunc(HWND, UINT, WPARAM, LPARAM)
.text:0000000140001110 DialogFunc proc near ; DATA XREF: start+Fl0
.text:0000000140001110 ; .rdata:00000001400020F0↓o ...
.text:0000000140001110
.text:0000000140001110 String= byte ptr -58h
.text:0000000140001110 arg_0= qword ptr 8
.text:0000000140001110
.v .text:0000000140001110 push rdi
.text:0000000140001112 sub rsp, 70h
.text:0000000140001116 sub edx, 10h
.text:0000000140001119 mov rdi, rcx
.text:000000014000111C jz loc_140001225
.text:0000000140001122 sub edx, 100h
.text:0000000140001128 jz loc_1400011B9
.text:000000014000112E cmp edx, 1
.text:0000000140001131 jnz short loc_14000113B
.text:0000000140001133 cmp r8w, 3E9h
.text:0000000140001139 jz short loc_140001143
.text:000000014000113B
.text:000000014000113B loc_14000113B: ; CODE XREF: DialogFunc+21↑j
.text:000000014000113B xor eax, eax
.text:000000014000113D add rsp, 70h
.text:0000000140001141 pop rdi
.text:0000000140001142 retn
.text:0000000140001143 ;
+evt::aaaaaaaa140001143

```

Тут алгоритм счёта примерно такой (исправьте, где я неправильно написал):

Тут используются 4 ASCII числа, они все будут прибавлены по одному разу и будут отниматься от 0C0, но из-за того, что 0C0 не является числом — поэтому это значение можно проигнорировать, состоит из 19 знаков ASCII: XXXX-XXXX-XXXX-XXXX. Используется шестнадцатеричная система.

Примерно такое вычисление: 30h+31h+32h+33h.