13.01.2024

Курс:

Практическая работа к уроку № Lesson_9

--
Реверс-инжиниринг программ с архитектурой x64

Задание:

Дана программа task-6. Необходимо получить ключ для вашего имени, который успешно примет программа.

Пример запуска программы:
task-6.exe

```
Name: <your name>
License Key: <license key>
```

Основные преимущества архитектуры x64 перед x86:

- 64-битное адресное пространство
- расширенный набор регистров
- обратная совместимость с x86
  Адресное пространство, по сравнению с архитектурой x86, стало сильно больше. 64-битный процессор теоретически может адресовать 16 экзабайт памяти (это 2^64). Но на
  практике, Windows с архитектурой x64 в настоящий момент поддерживает только 16 ТБ (это 2^44). Основное ограничение связано с тем, что современные процессоры могут обеспечивать доступ лишь к 1 терабайту физической памяти (это 2^40).

Данная архитектура представляет собой расширение архитектуры x86 с полной обратной
совместимостью. Существует множество вариантов названия данной архитектуры: AMD64,
x86-64, x64 и можно еще найти несколько. Будем называть ее просто x64.

Как и в Windows с архитектурой x86, адресуемый диапазон памяти делится на пользовательские адреса и на системные. Каждый процесс получает 8 ТБ памяти и 8

ТБ остается для ядра ОС (напомню, что в Windows с архитектурой x86 было выделено 2 ГБ для
процесса и 2ГБ для ядра ОС).
Размер страницы в памяти также составляет 4 КБ. Первые 64 КБ адресного пространства
никогда не отображаются, т.е. наименьший правильный адрес это 0x10000. Также стоить знать, что в Windows с архитектурой x64 системные DLL загружаются в память по адресу, который всегда выше 4 ГБ.

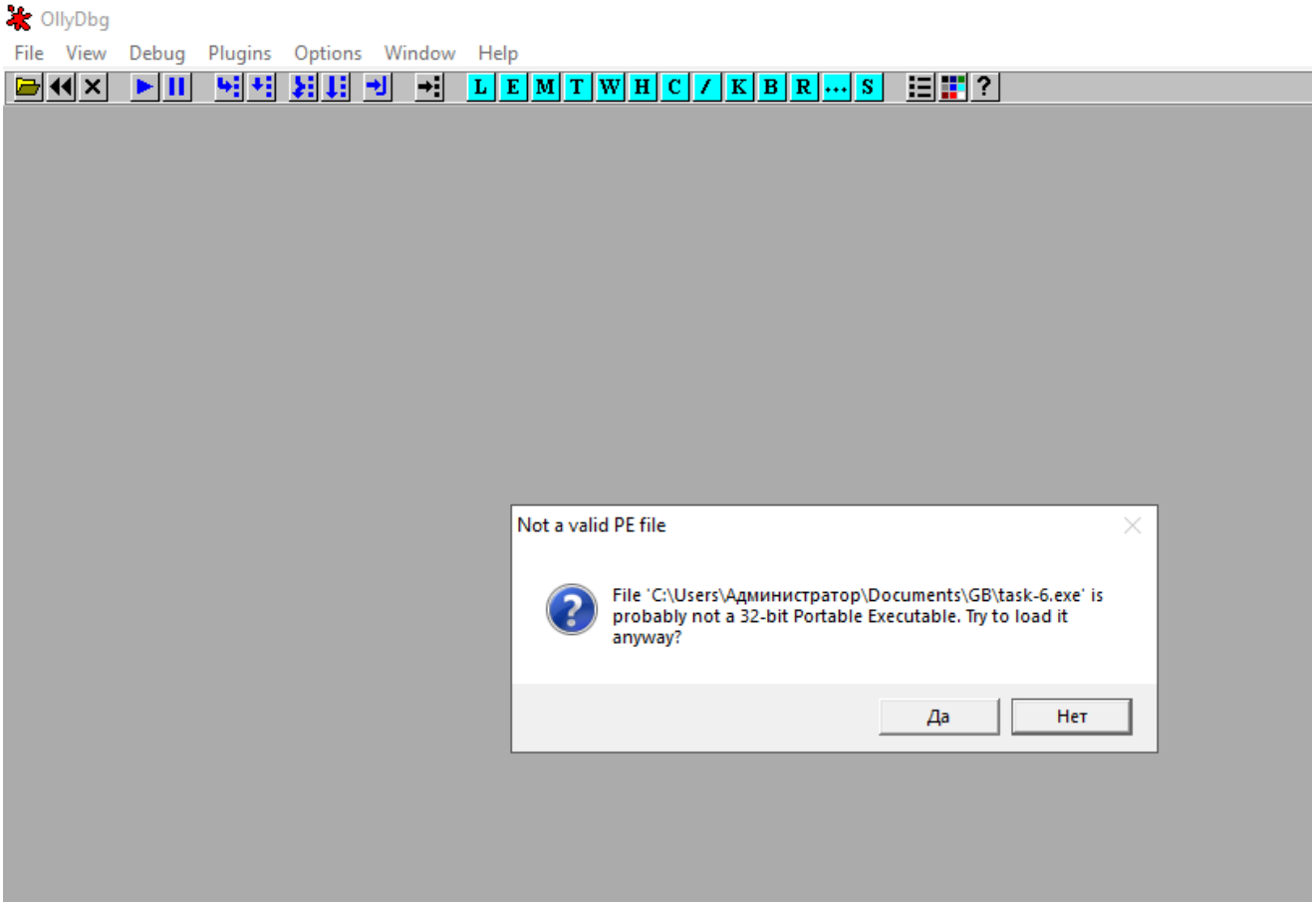
Запускаем в CMD *task-6.exe*

Insert name: Andrew
License key: aaaabbbbccccdddd

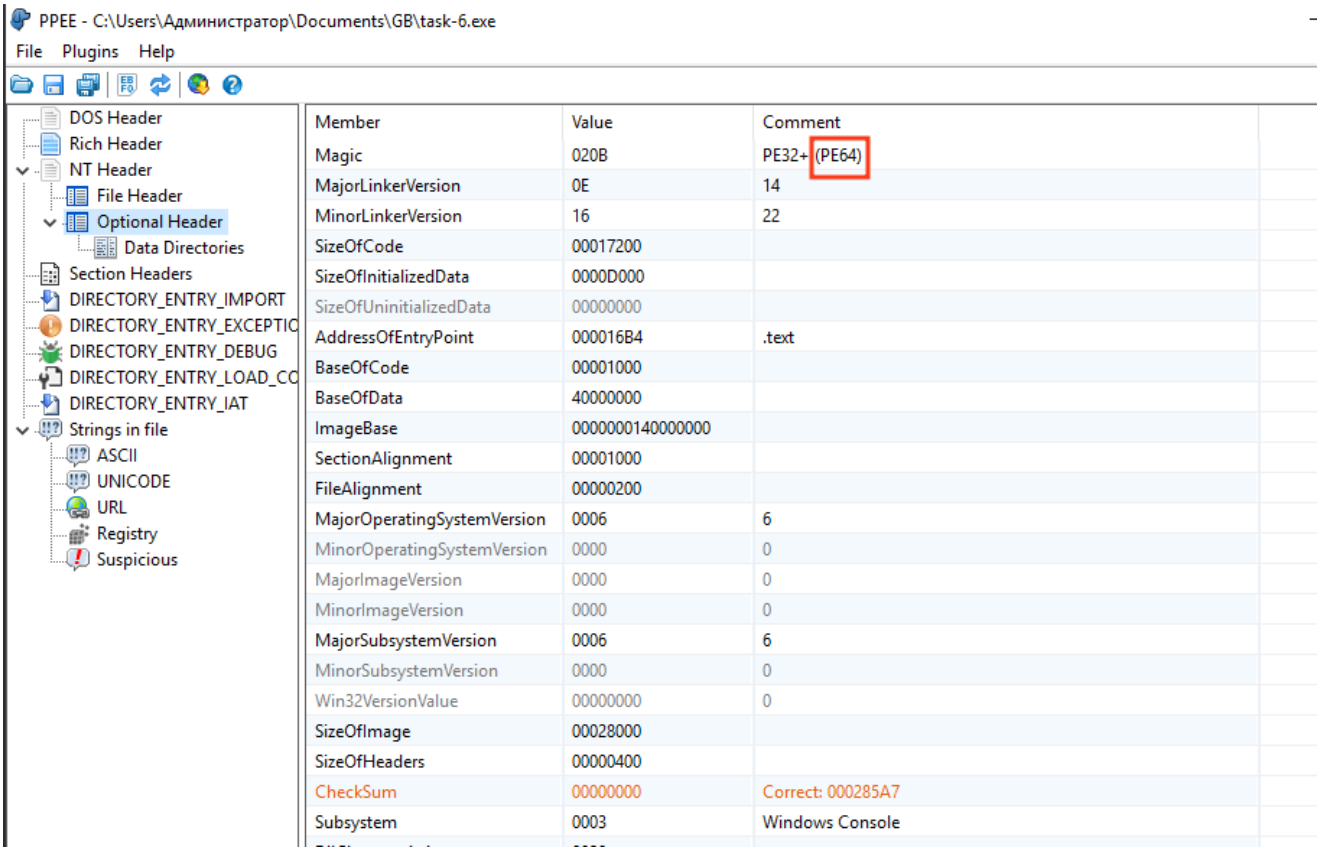
```
C:\Users\Администратор\Documents\GB>task-6.exe

Name: Andrew

License Key: aaaabbbbccccdddd

ERROR. Your key is incorrect.
```

```
C:\Users\Администратор\Documents\GB>task-6.exe

Name: Andrew

License Key: qwerty

ERROR. Your key is incorrect.
```

Открываем *task-6.exe* под отладчиком в OllyDbg

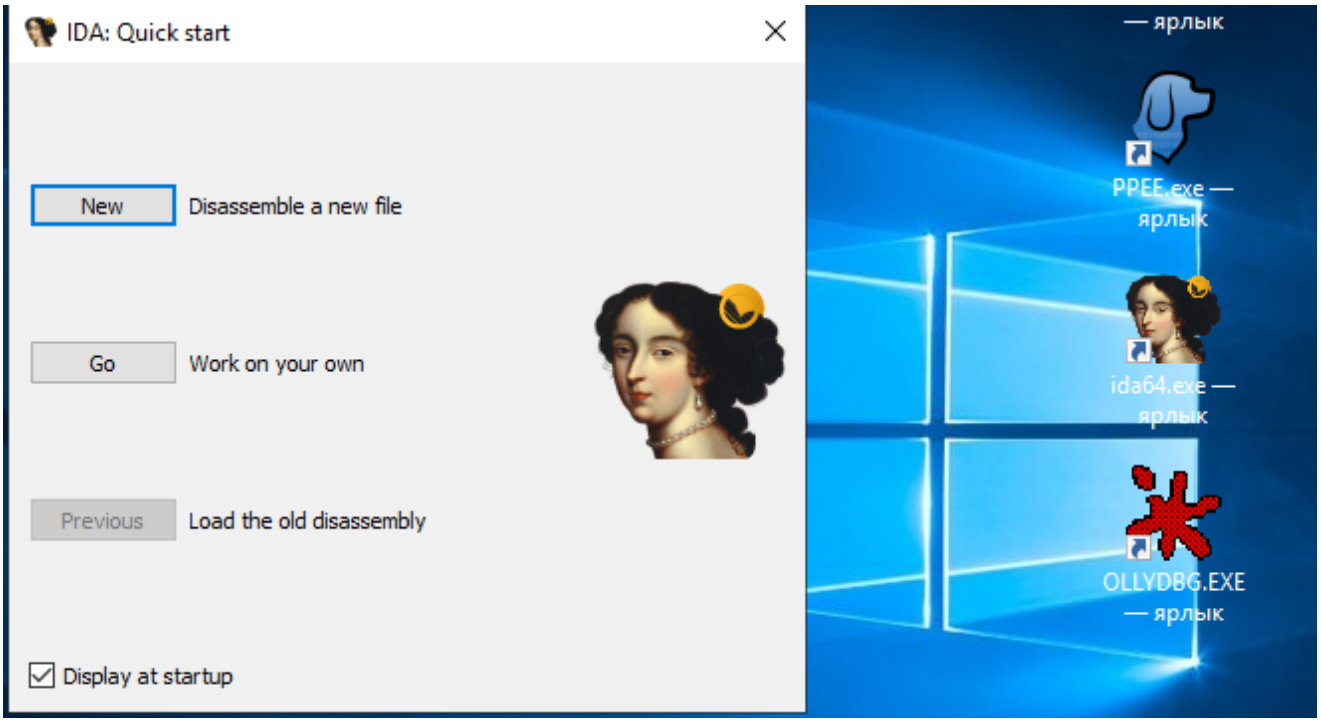


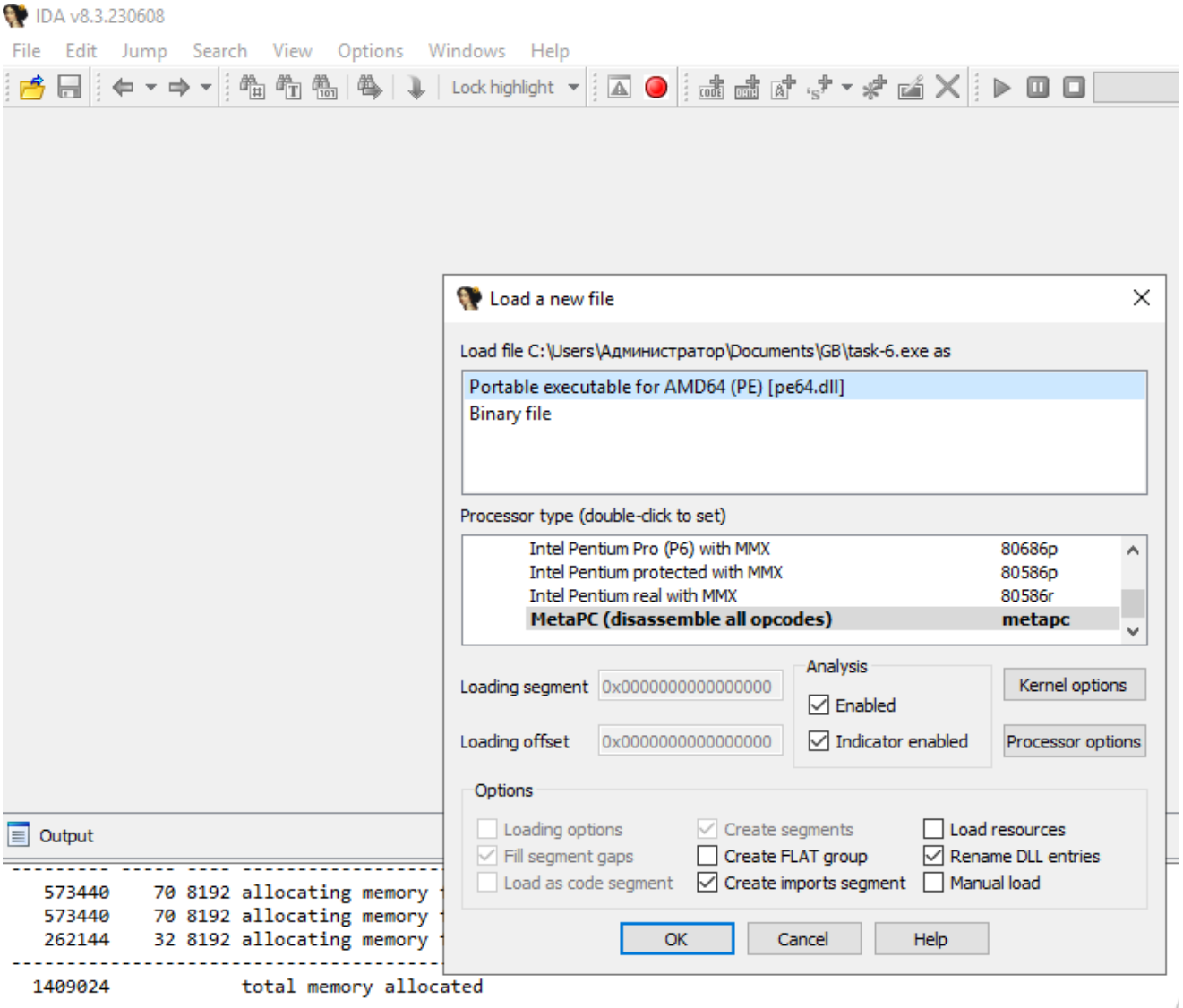
Открываем *task-6.exe* в PPEE

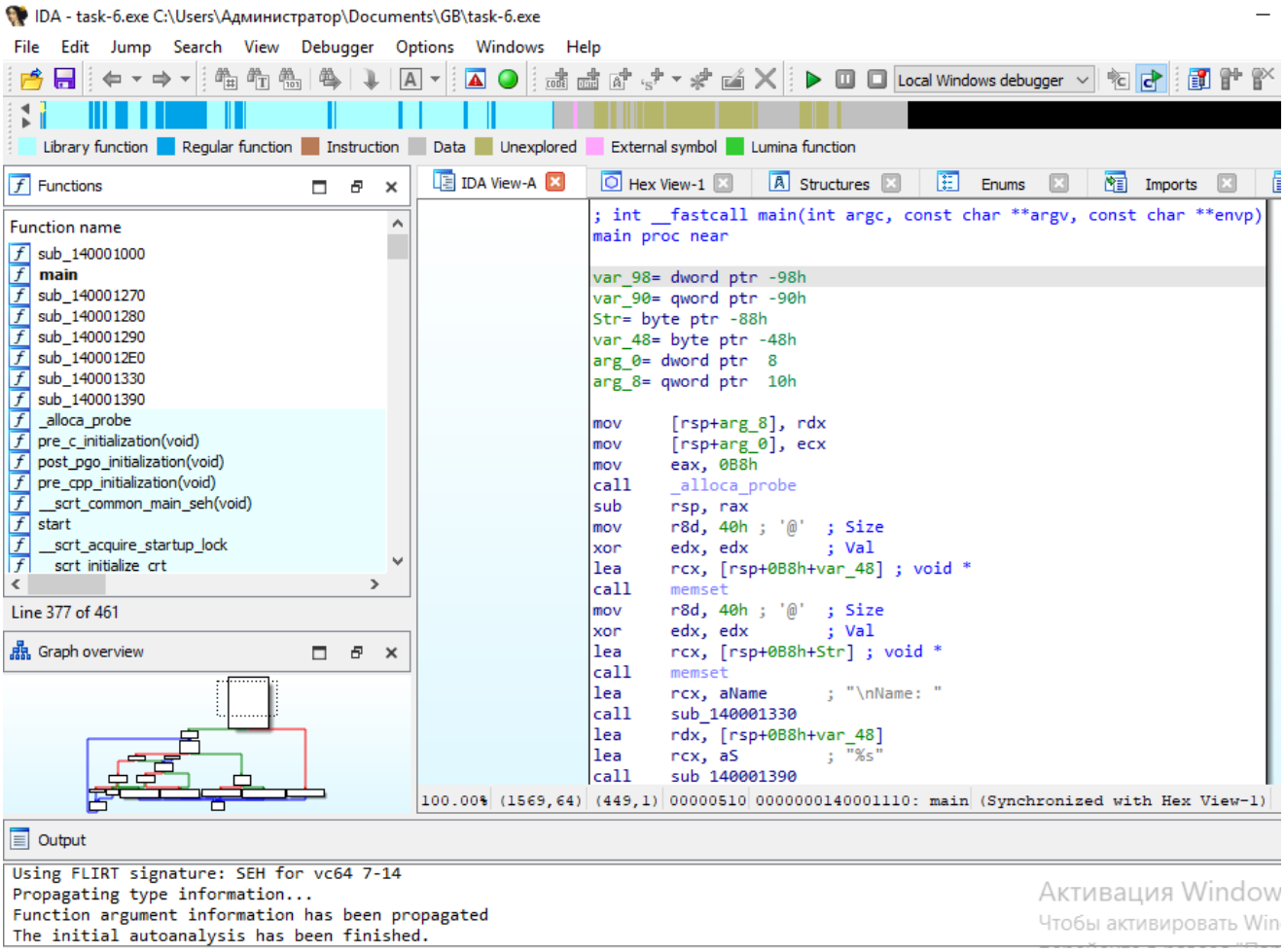


Видим, что архитектура x64.

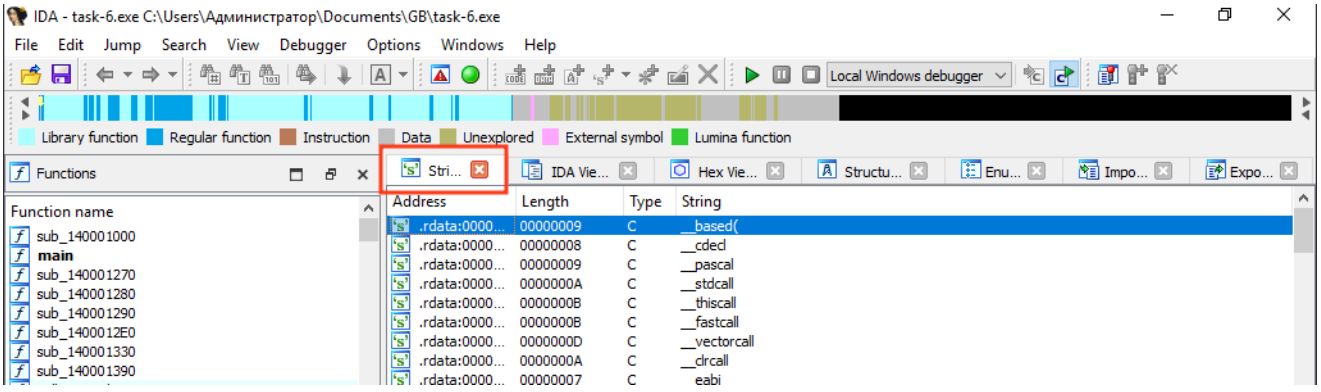
Открываем *task-6.exe* в PPEE

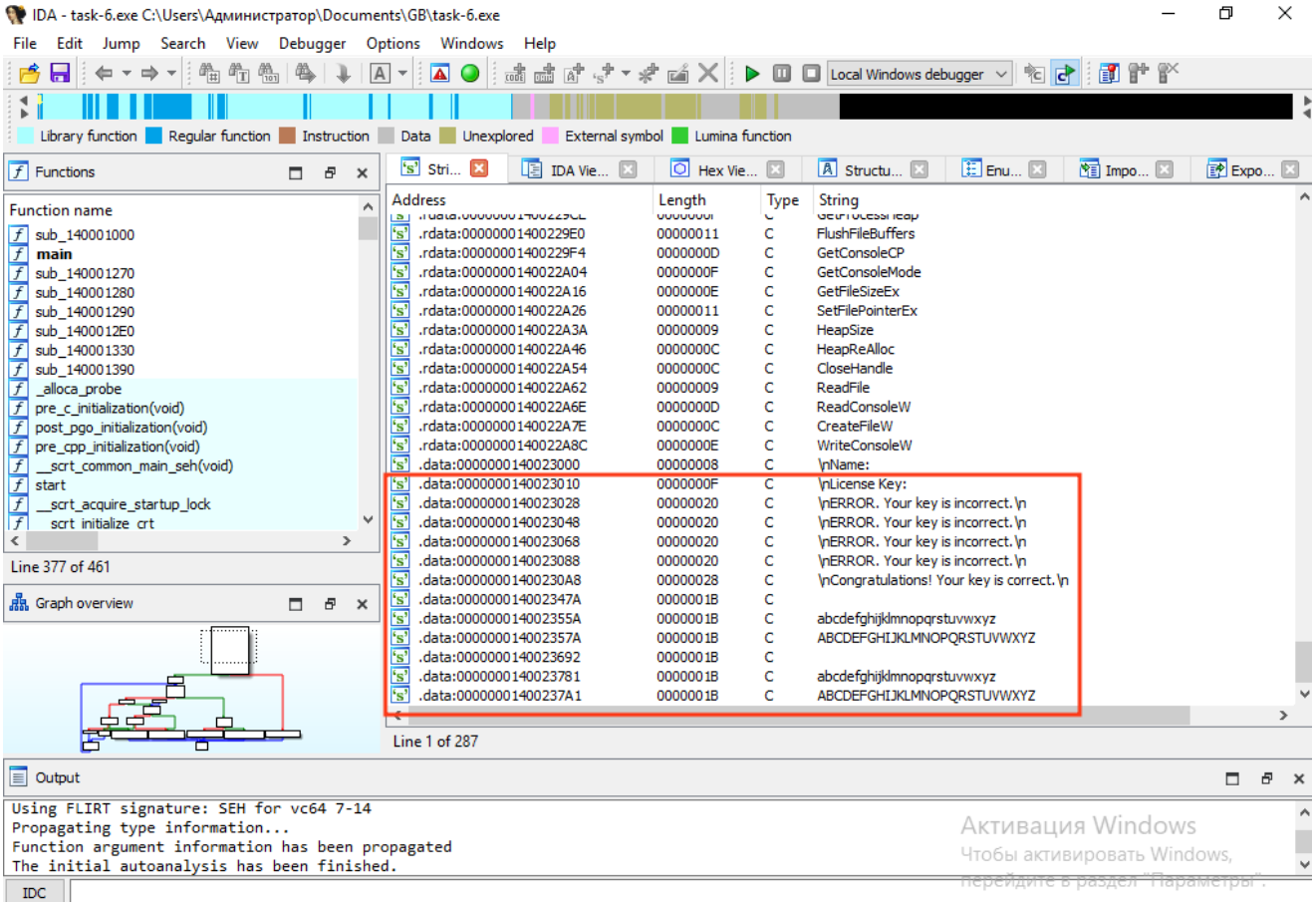

IDA: Quick start
New
Disassemble a new file
Go
Work on your own
Previous
Load the old disassembly
Display at startup
— ярлык
PPEE.exe — ярлык
ida64.exe — ярлык
OLLYDBG.EXE — ярлык

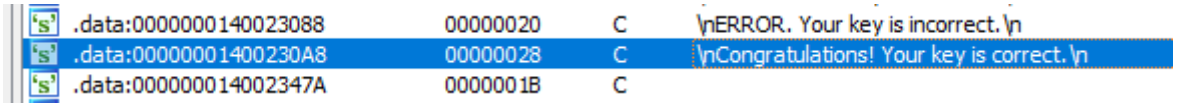

IDA v8.3.230608
File Edit Jump Search View Options Windows Help
Lock highlight
Load a new file
Load file C:\Users\Администратор\Documents\GB\task-6.exe as
Portable executable for AMD64 (PE) [pe64.dll]
Binary file
Processor type (double-click to set)
Intel Pentium Pro (P6) with MMX 80686p
Intel Pentium protected with MMX 80586p
Intel Pentium real with MMX 80586r
MetaPC (disassemble all opcodes) metapc
Loading segment 0x0000000000000000
Loading offset 0x0000000000000000
Analysis
Enabled
Indicator enabled
Kernel options
Processor options
Options
Loading options
Fill segment gaps
Load as code segment
Create segments
Create FLAT group
Create imports segment
Load resources
Rename DLL entries
Manual load
OK
Cancel
Help
Output
573440 70 8192 allocating memory
573440 70 8192 allocating memory
262144 32 8192 allocating memory
1409024 total memory allocated
Loading processor module C:\Program Files\ReverseIng\IDA Freeware 8.3\procs\pc64.dll for metapc...Initiali



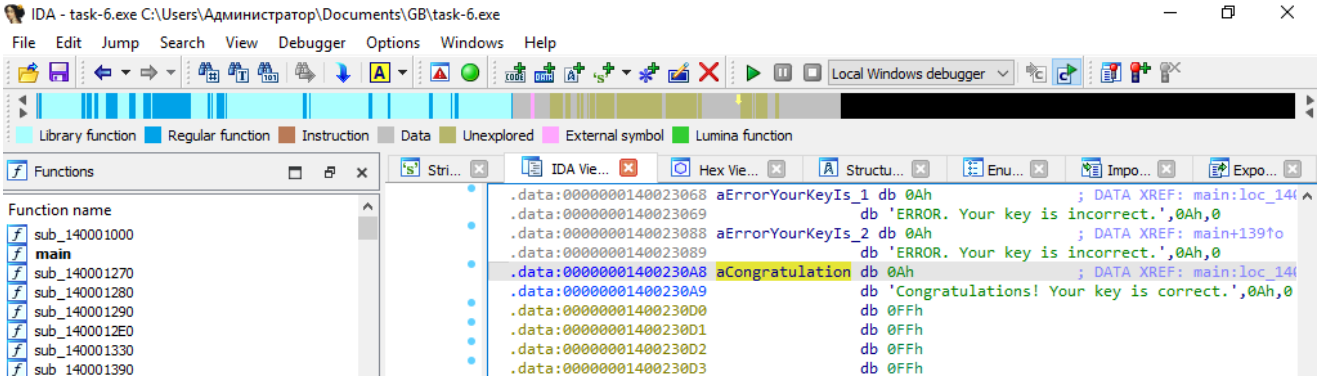
View -> Open subviews -> Strings Schift + F12





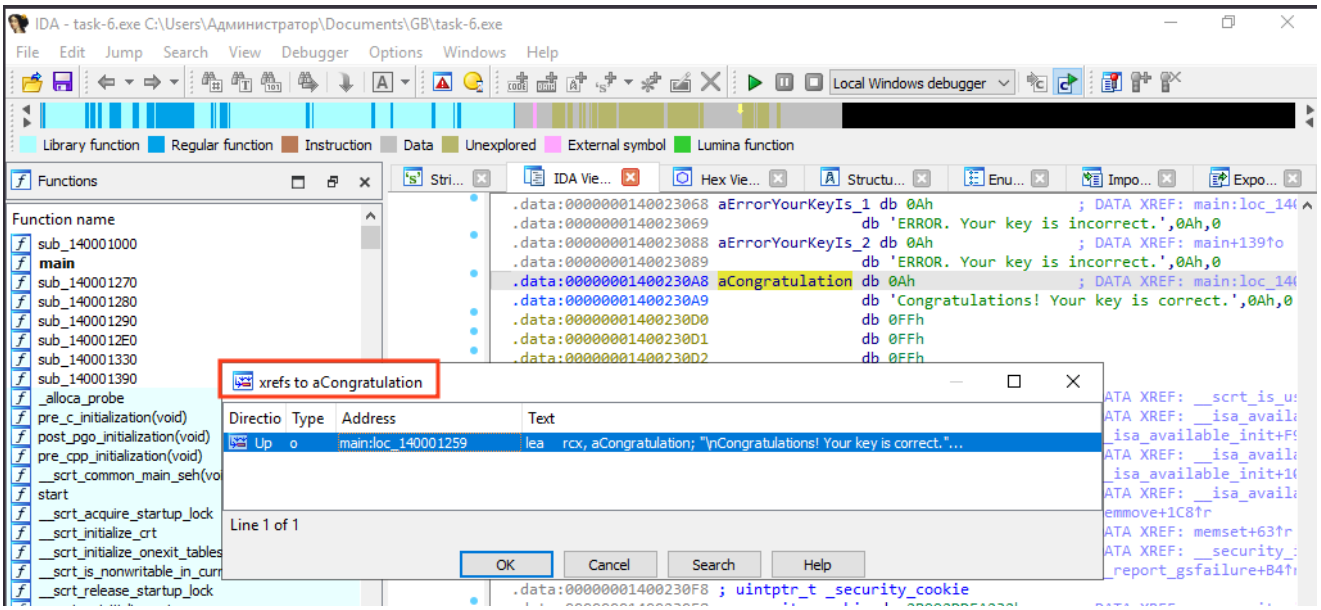


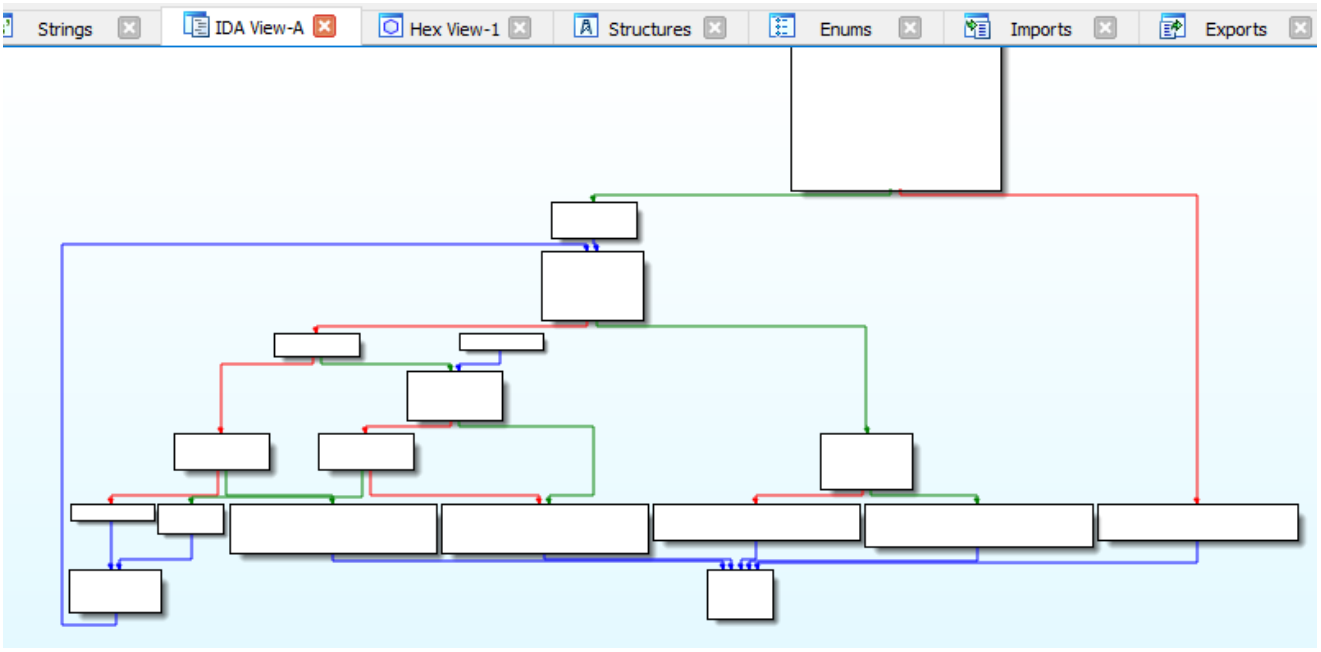
Переходим в секцию данных:

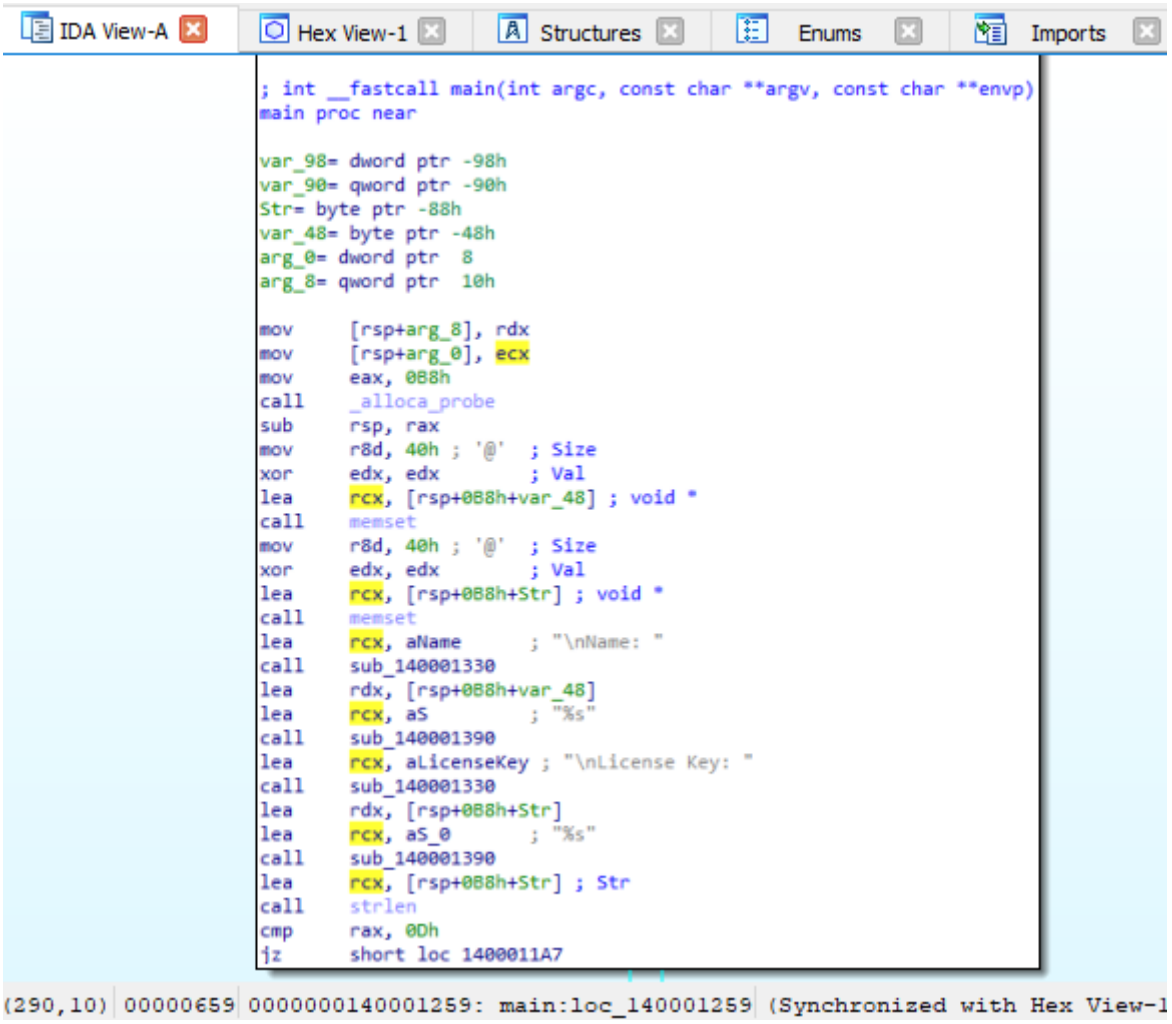


Посредством перекрестных ссылок попадаем:

X (xrefs to aCongratulation)


IDA - task-6.exe C:\Users\Администратор\Documents\GB\task-6.exe
File Edit Jump Search View Debugger Options Windows Help
Local Windows debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
sub_140001000
main
sub_140001270
sub_140001280
sub_140001290
sub_1400012E0
sub_140001330
sub_140001390
_alloca_probe
pre_c_initialization(void)
post_pgo_initialization(void)
pre_cpp_initialization(void)
__scrt_common_main_seh(vo
start
__scrt_acquire_startup_lock
__scrt_initialize_crt
__scrt_initialize_onexit_tables
__scrt_is_nonwritable_in_curr
__scrt_release_startup_lock
Stri...
IDA Vie...
Hex Vie...
Structu...
Enu...
Impo...
Expo...
.data:0000000140023068 aErrorYourKeyIs_1 db 0Ah ; DATA XREF: main:loc_140
.data:0000000140023069 db 'ERROR. Your key is incorrect.',0Ah,0
.data:0000000140023088 aErrorYourKeyIs_2 db 0Ah ; DATA XREF: main+139↑o
.data:0000000140023089 db 'ERROR. Your key is incorrect.',0Ah,0
.data:00000001400230A8 aCongratulation db 0Ah ; DATA XREF: main:loc_140
.data:00000001400230A9 db 'Congratulations! Your key is correct.',0Ah,0
.data:00000001400230D0 db 0FFh
.data:00000001400230D1 db 0FFh
.data:00000001400230D2 db 0FFh
xrefs to aCongratulation
Directio Type Address Text
Up o main:loc_140001259 lea rcx, aCongratulation; "\nCongratulations! Your key is correct."...
Line 1 of 1
OK
Cancel
Search
Help
.data:00000001400230F8 ; uintptr_t _security_cookie


Strings
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports



Похожа на функцию main

Функция имеет 3 локальные переменные и 2 входных аргумента

```
var_98= dword ptr -98h
var_90= qword ptr -90h
Str= byte ptr -88h
var_48= byte ptr -48h
arg_0= dword ptr  8
arg_8= qword ptr  10h
```

Создание локальных переменных

```
var_48= byte ptr -48h
arg_0= dword ptr  8
arg_8= qword ptr  10h

mov     [rsp+arg_8], rdx
mov     [rsp+arg_0], ecx
mov     eax, 088h
call    _alloca_probe
sub     rsp, rax
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+var_48] ; void *
call    memset
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+Str] ; void *
call    memset
lea     rcx, aName      ; "\nName: "
call    sub_140001330
lea     rdx, [rsp+088h+var_48]
lea     rcx, aS         ; "%s"
call    sub_140001390
lea     rcx, aLicenseKey ; "\nLicense Key: "
call    sub_140001330
```

Вызов одинаковых функций

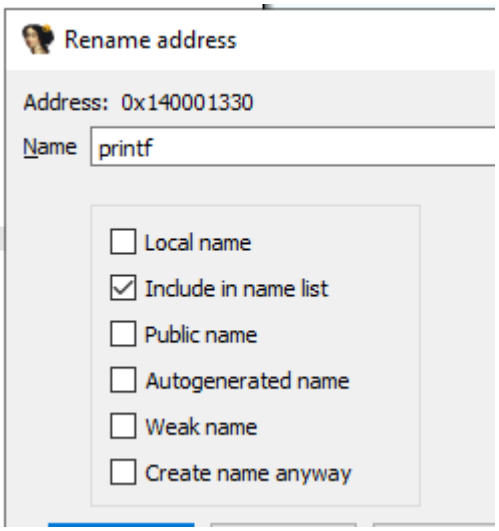
```
mov     [rsp+arg_8], rdx
mov     [rsp+arg_0], ecx
mov     eax, 088h
call    _alloca_probe
sub     rsp, rax
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+var_48] ; void *
call    memset
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+Str] ; void *
call    memset
lea     rcx, aName      ; "\nName: "
call    sub_140001330
lea     rdx, [rsp+088h+var_48]
lea     rcx, aS         ; "%s"
```

- memset:
  d нее передаются 0x40 и 0
  Edx положит значение 0 в Edx
  var_40 указатель на локальную переменную

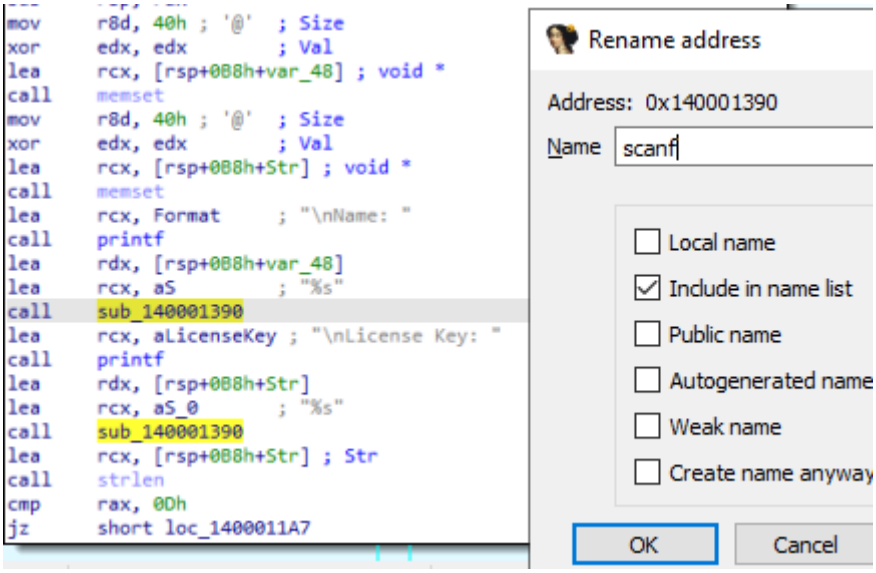
Далее

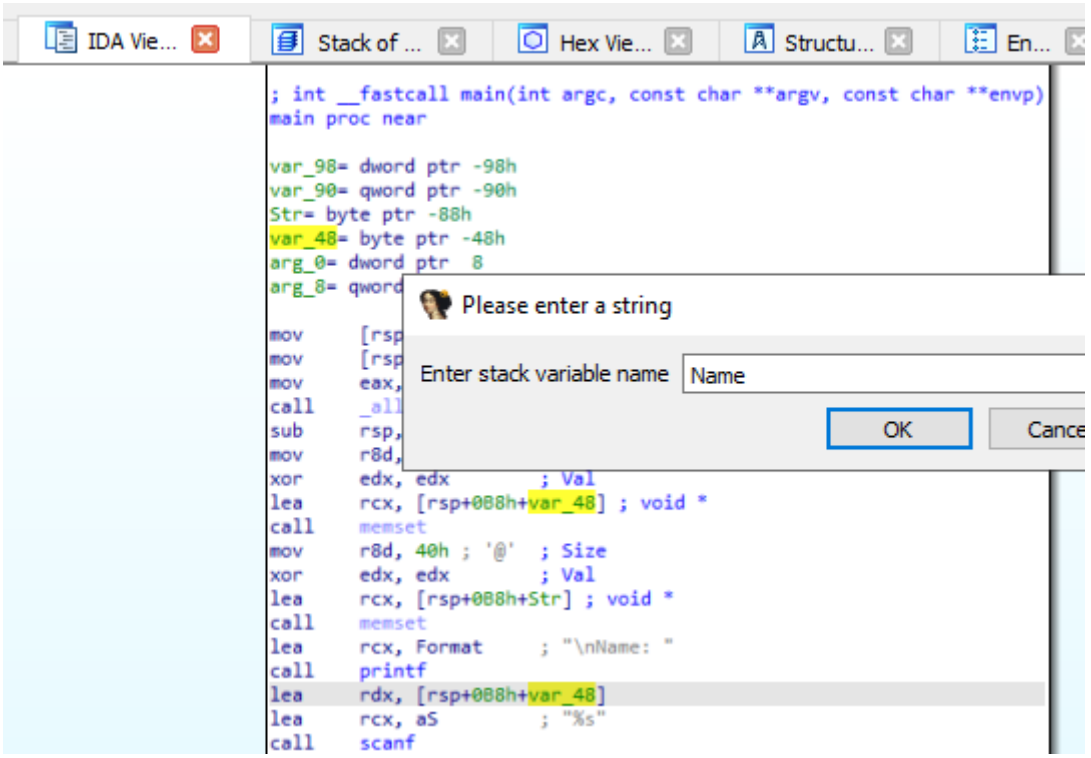
- memset. sub_140001330 переименуем в prinftf

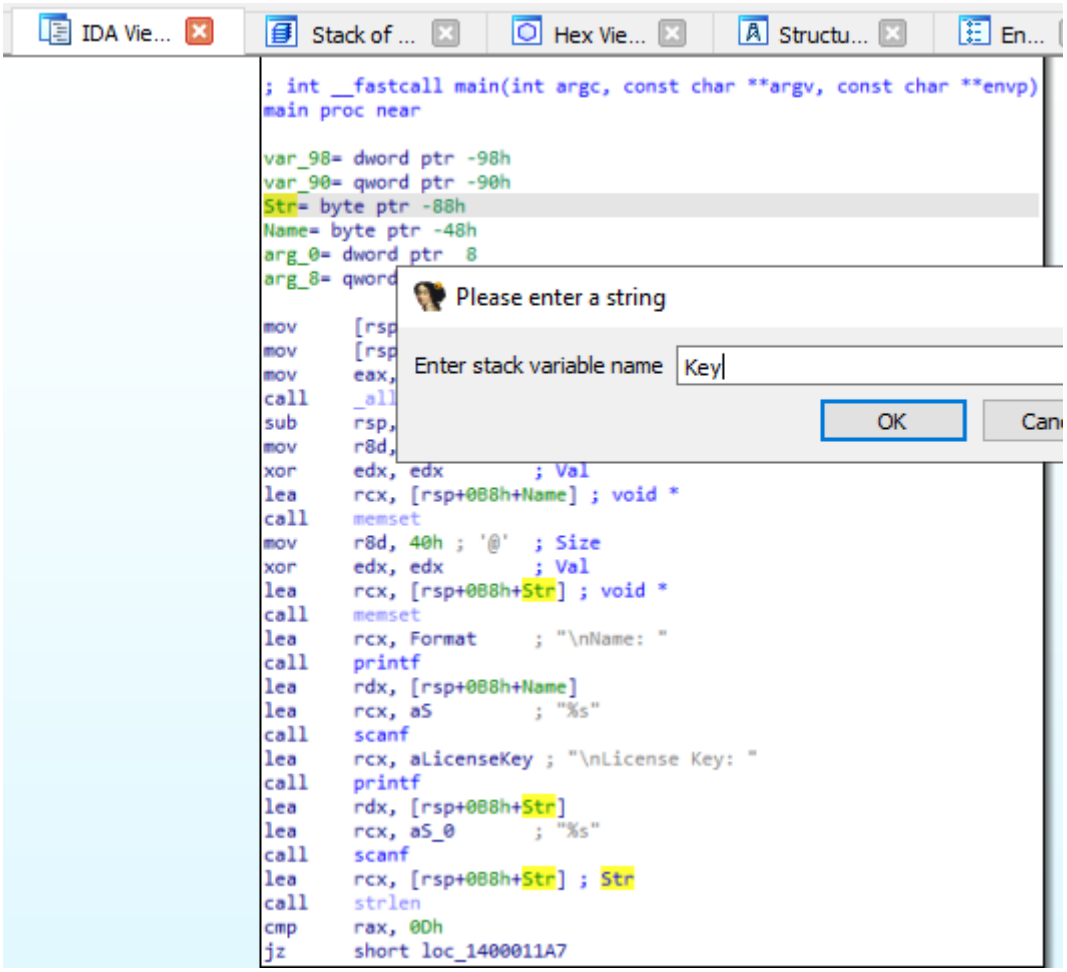
```
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+var_48] ; void *
call    memset
mov     r8d, 40h ; '@'  ; Size
xor     edx, edx        ; Val
lea     rcx, [rsp+088h+Str] ; void *
call    memset
lea     rcx, aName      ; "\nName: "
call    sub_140001330
lea     rdx, [rsp+088h+var_48]
lea     rcx, aS         ; "%s"
call    sub_140001390
lea     rcx, aLicenseKey ; "\nLicense Key: "
call    sub_140001330
lea     rdx, [rsp+088h+Str]
lea     rcx, aS_0       ; "%s"
call    sub_140001390
lea     rcx, [rsp+088h+Str] ; Str
call    strlen
cmp     rax, 0Dh
```

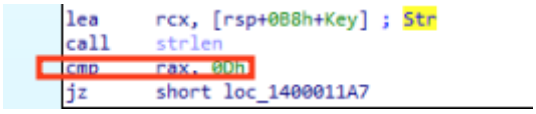


Форматная строка:
Считывается. Переименуем в scanf


mov r8d, 40h ; '@' ; Size
xor edx, edx ; Val
lea rcx, [rsp+088h+var_48] ; void *
call memset
mov r8d, 40h ; '@' ; Size
xor edx, edx ; Val
lea rcx, [rsp+088h+Str] ; void *
call memset
lea rcx, Format ; "\nName: "
call printf
lea rdx, [rsp+088h+var_48]
lea rcx, aS ; "%s"
call sub_140001390
lea rcx, aLicenseKey ; "\nLicense Key: "
call printf
lea rdx, [rsp+088h+Str]
lea rcx, aS_0 ; "%s"
call sub_140001390
lea rcx, [rsp+088h+Str] ; Str
call strlen
cmp rax, 0Dh
jz short loc_1400011A7
Rename address
Address: 0x140001390
Name scanf
Local name
Include in name list
Public name
Autogenerated name
Weak name
Create name anyway
OK
Cancel

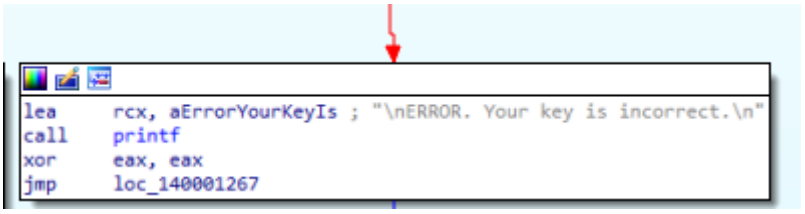

IDA Vie...
Stack of ...
Hex Vie...
Structu...
En...
; int __fastcall main(int argc, const char **argv, const char **envp)
main proc near
var_98= dword ptr -98h
var_90= qword ptr -90h
Str= byte ptr -88h
var_48= byte ptr -48h
arg_0= dword ptr 8
Please enter a string
Enter stack variable name
Name
OK
Cance
xor edx, edx ; Val
lea rcx, [rsp+088h+var_48] ; void *
call memset
mov r8d, 40h ; '@' ; Size
xor edx, edx ; Val
lea rcx, [rsp+088h+Str] ; void *
call memset
lea rcx, Format ; "\nName: "
call printf
lea rdx, [rsp+088h+var_48]
lea rcx, aS ; "%s"
call scanf



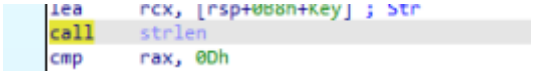


Сравнение ключа 0x0D

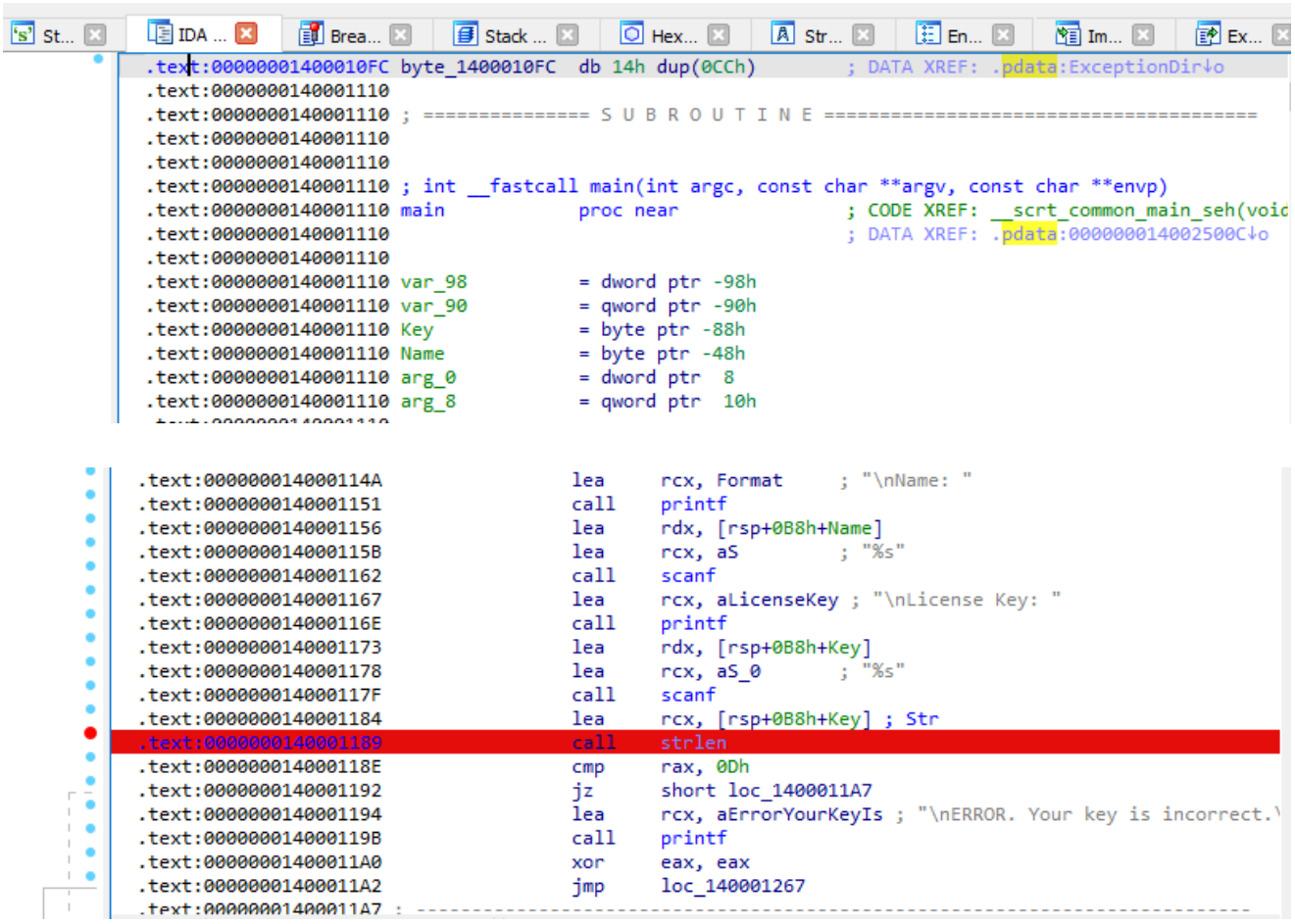
Если сравнение не прроходит, получаем Error



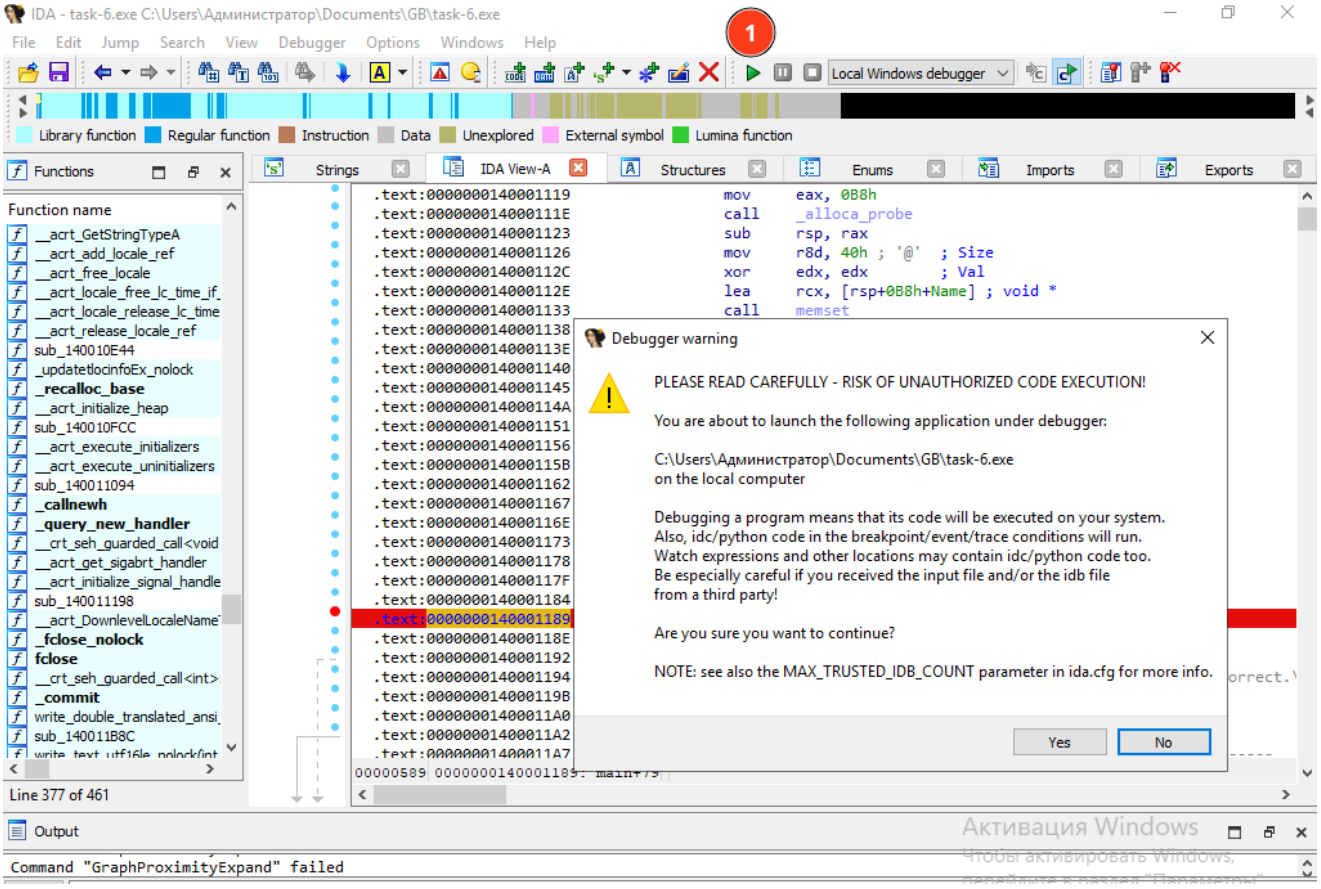
Ставим точку останова

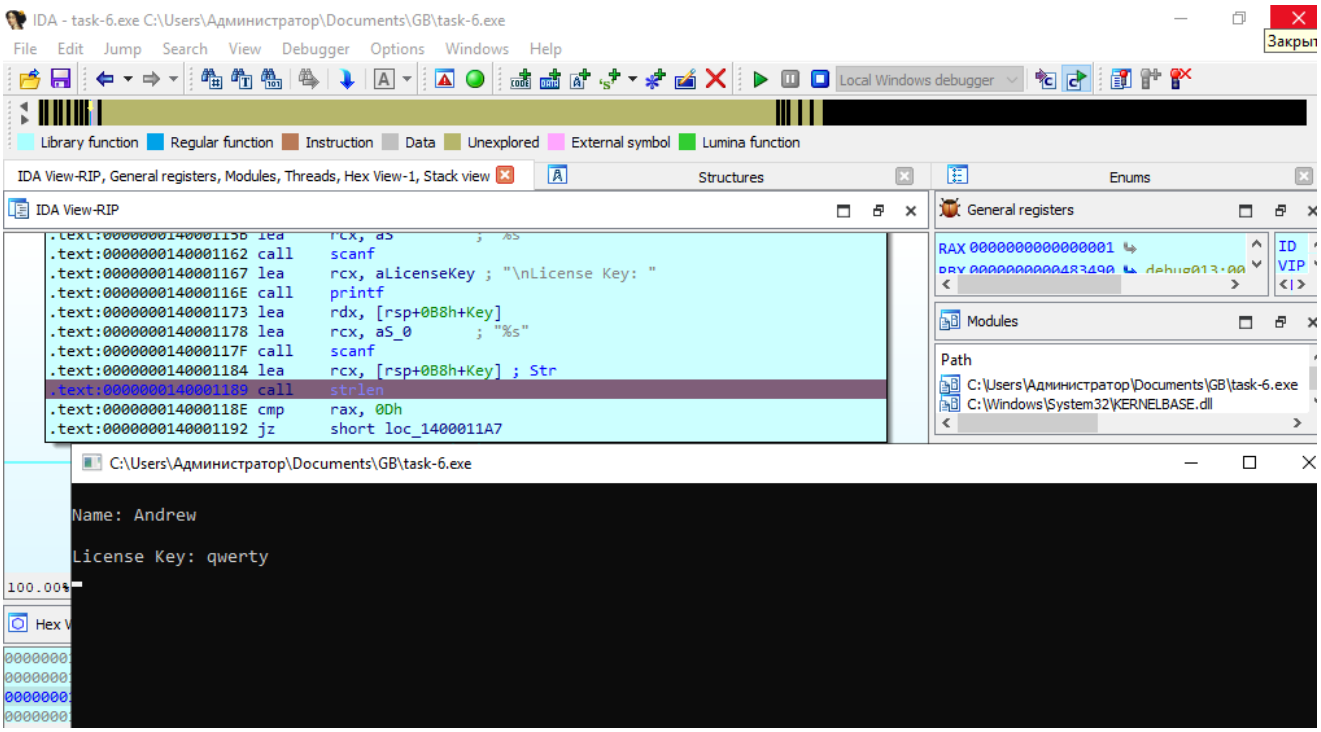


(правая мышь Test view)

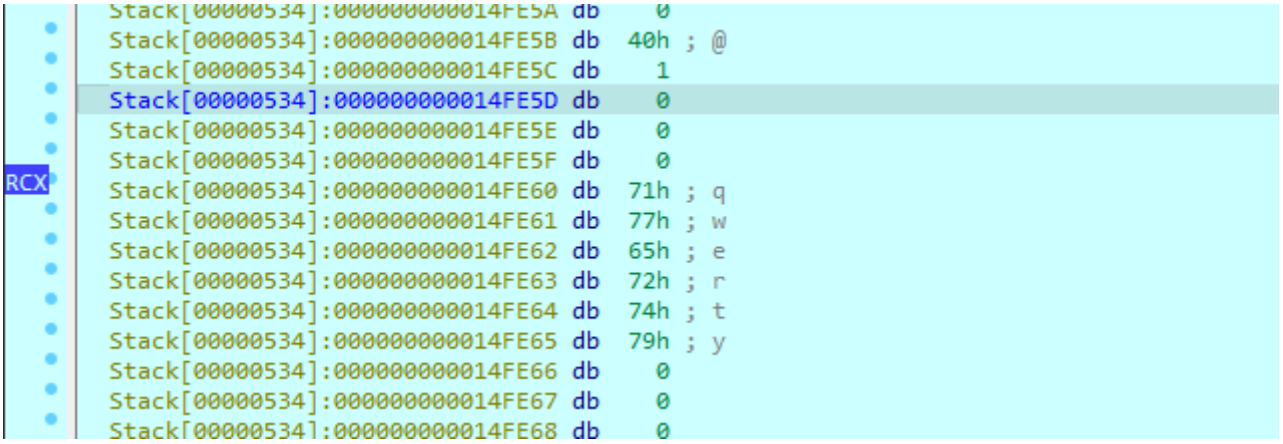


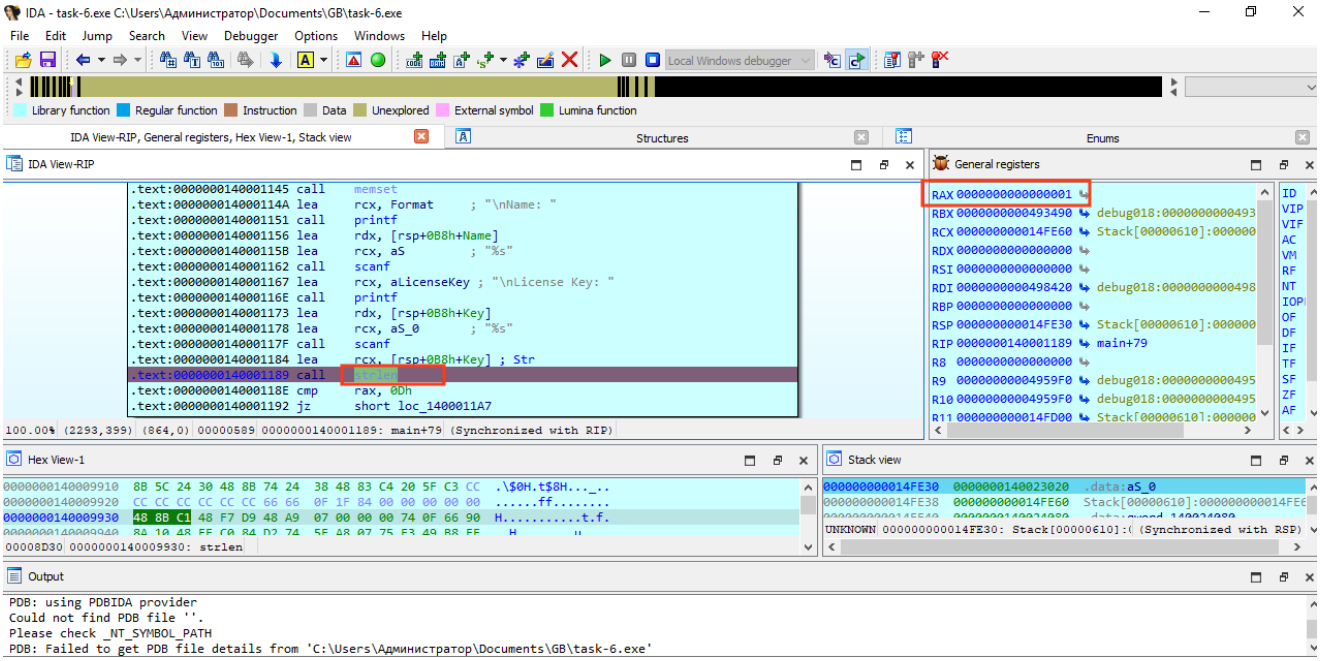
Запускаем программу





Key = qwerty



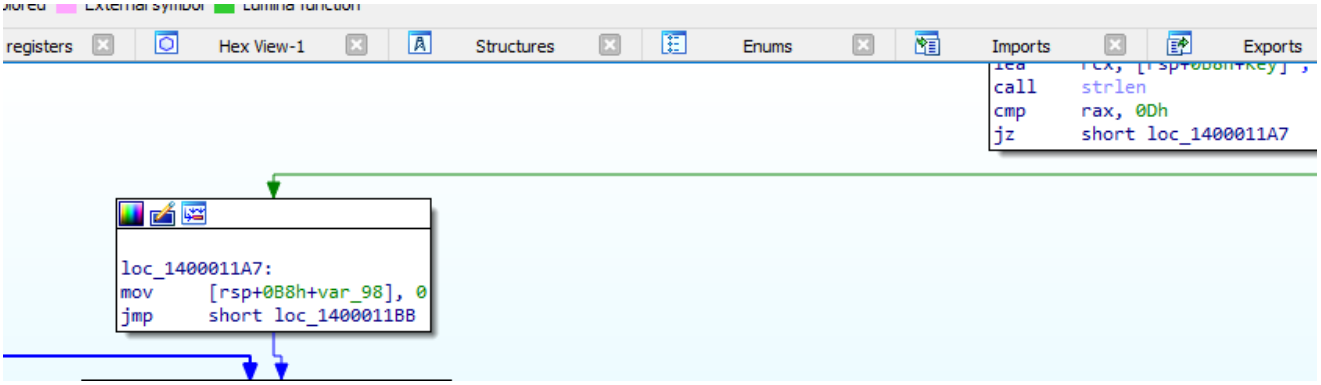


Выполним функцию и посмотрим регистр RAX

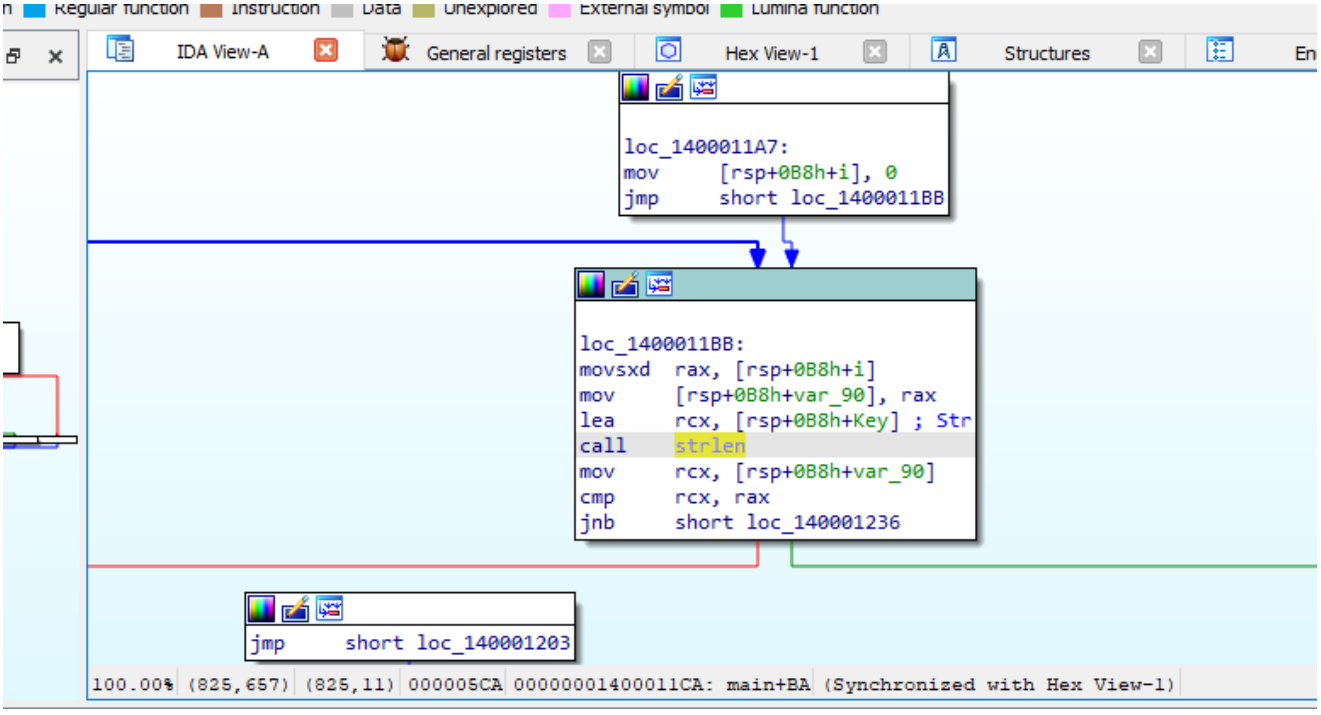
RAX 06

Переходим в режим Graph View

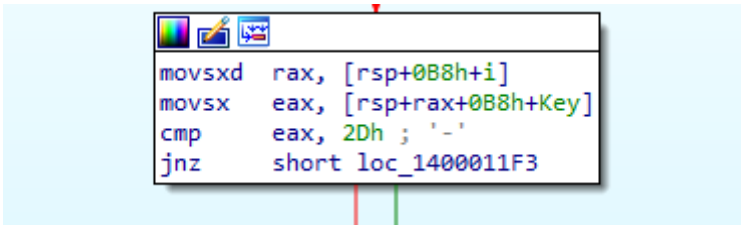
Просматриваем код. Видим цикл

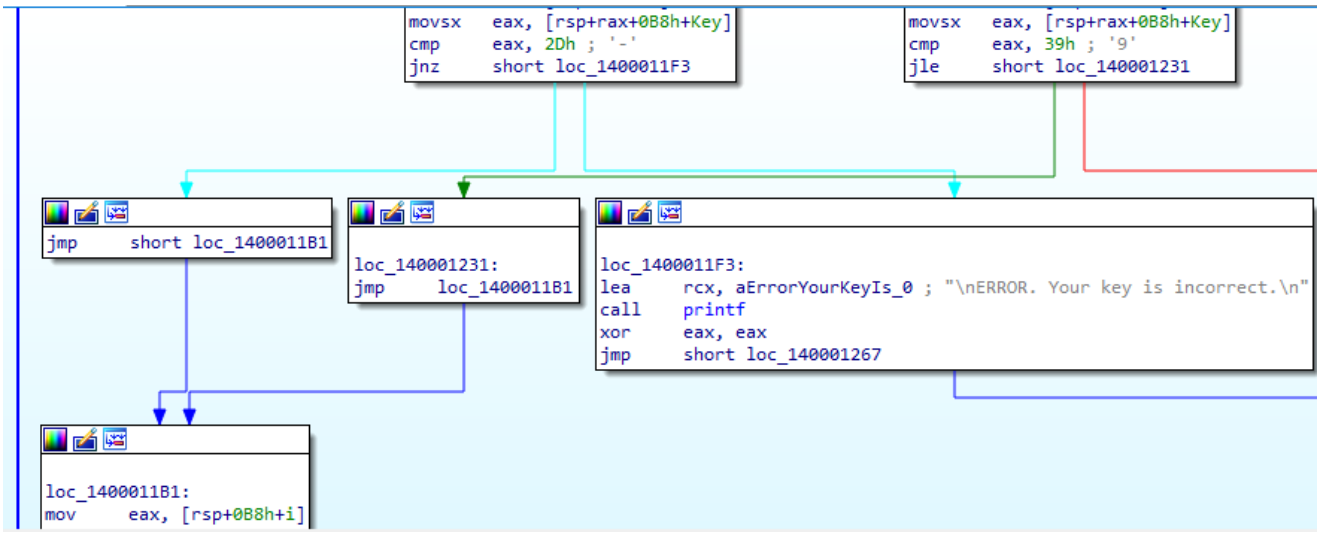


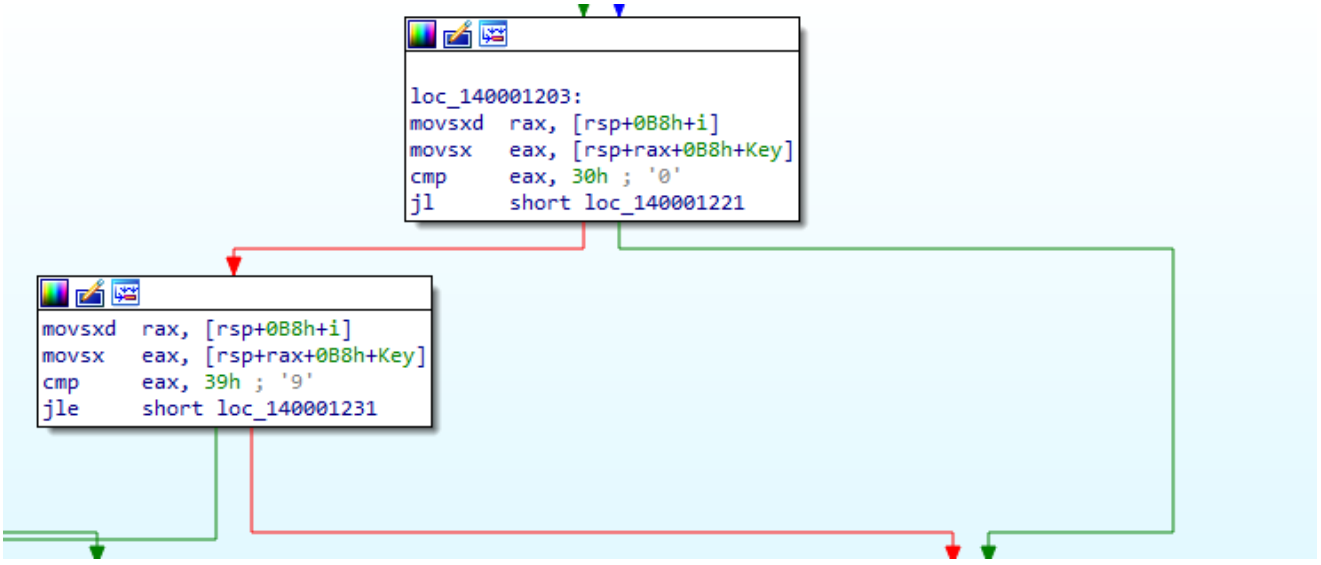
Переименовываем var_98 в i



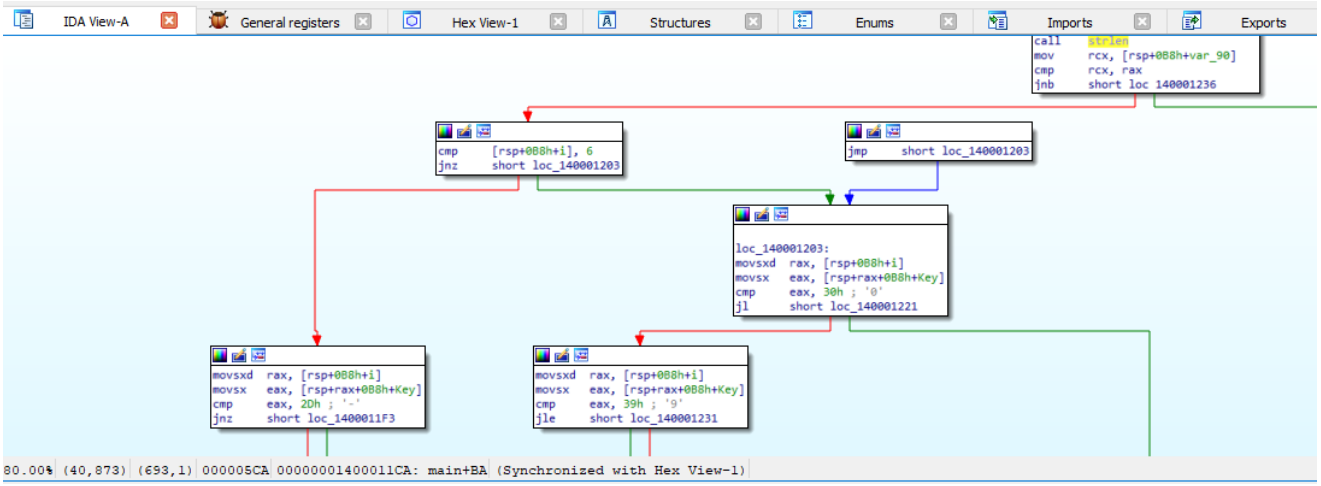
Условие выхода из цикла - сравнение с итератора с длиной строки Key
Длина 0x2D (2Dh) / 0xD = 13 / сравнивается в нашем случае с 6 (qwerty)
2D по таблице Asci равно - (тире)





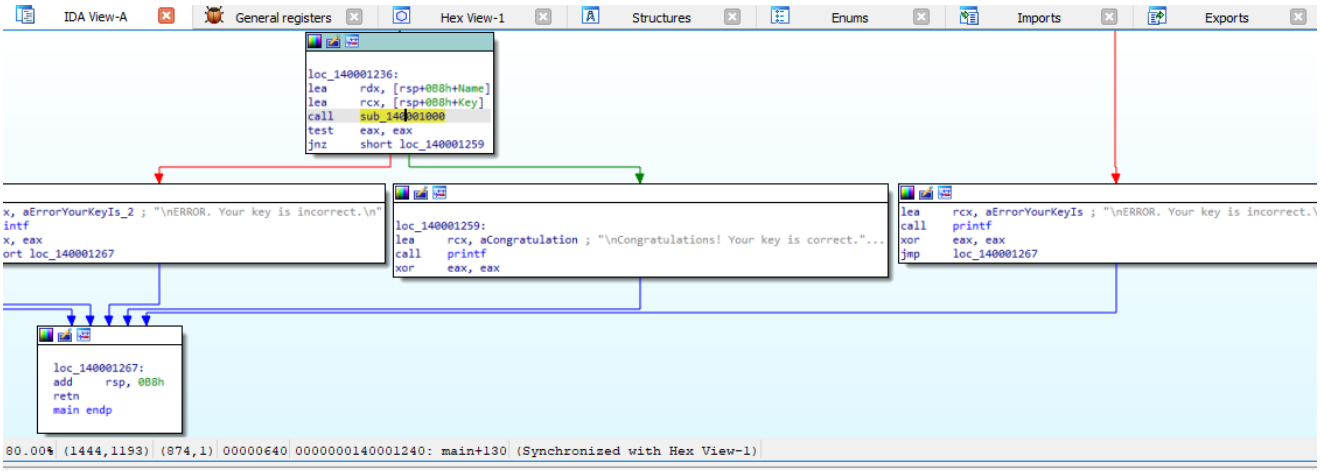


0x30 (30h) в таблице Asci равно 0
0x39 (39h) в таблице Asci равно 9



Т.е. условие выполнится, если

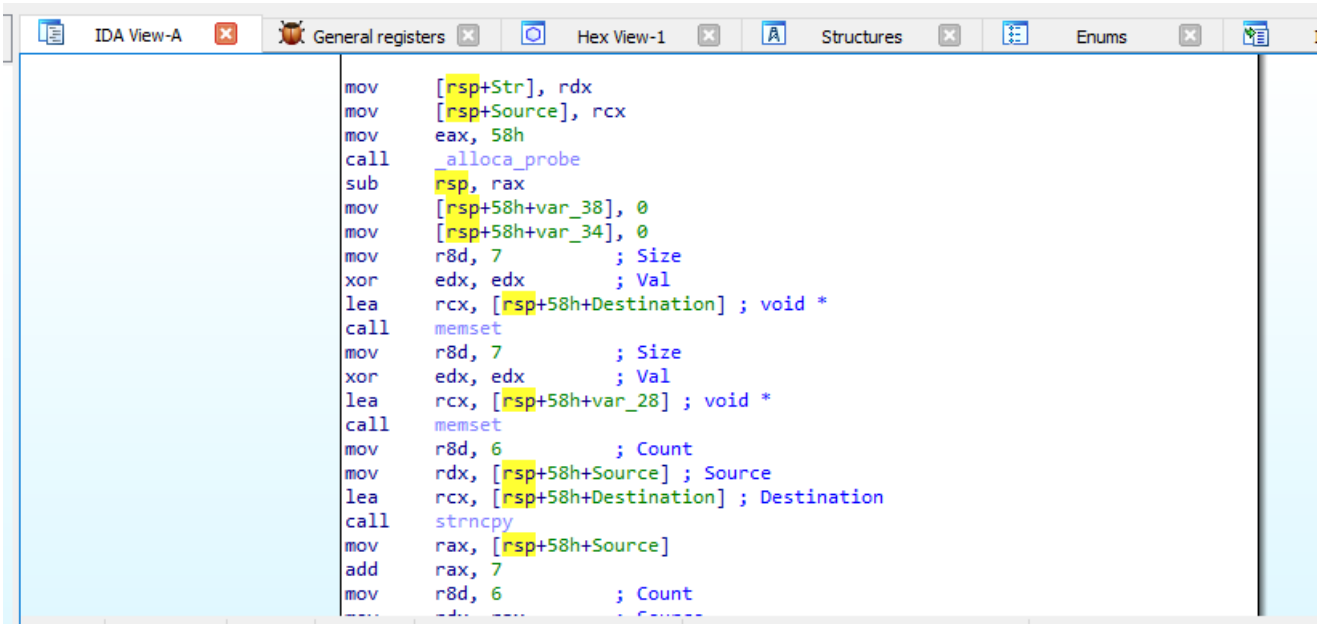
- на 6 позиции будет тире - (2Dh) cmp,
- а все остальные символы должны быть числами в диапазоне от 0x30 по 0x39 (0, 1, ..., 9)

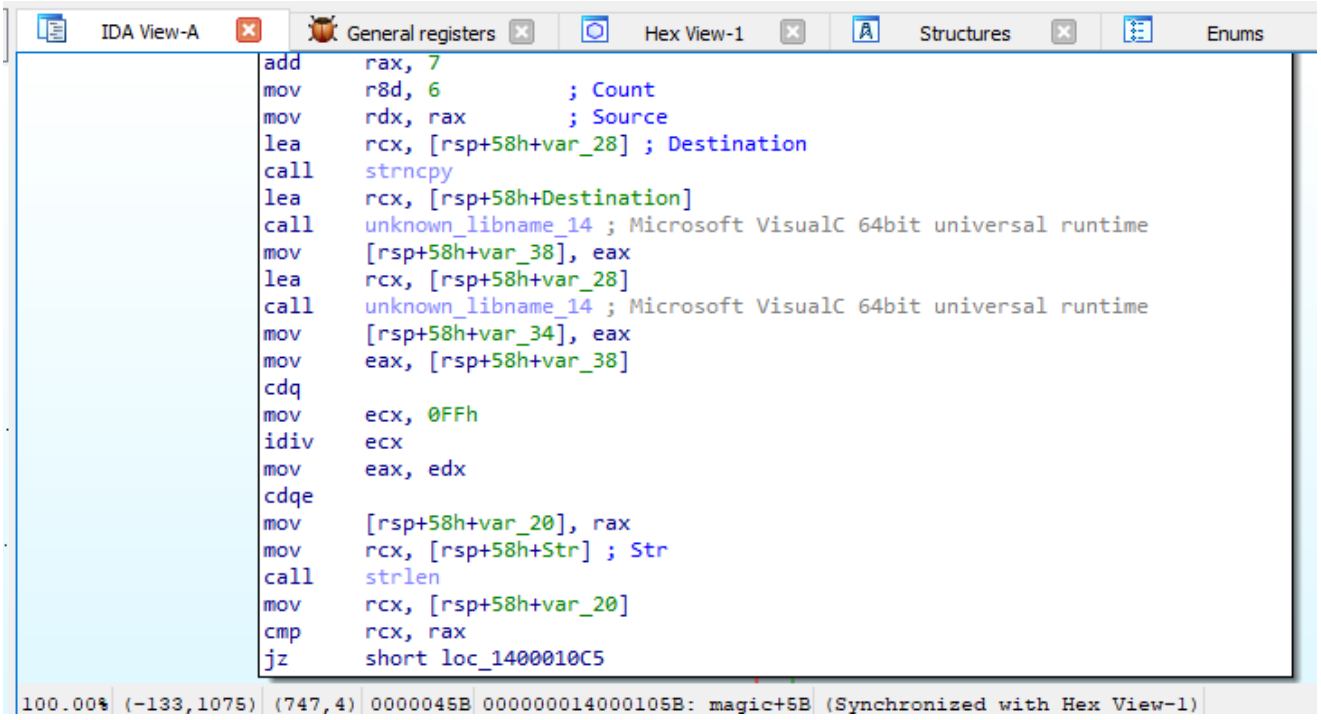


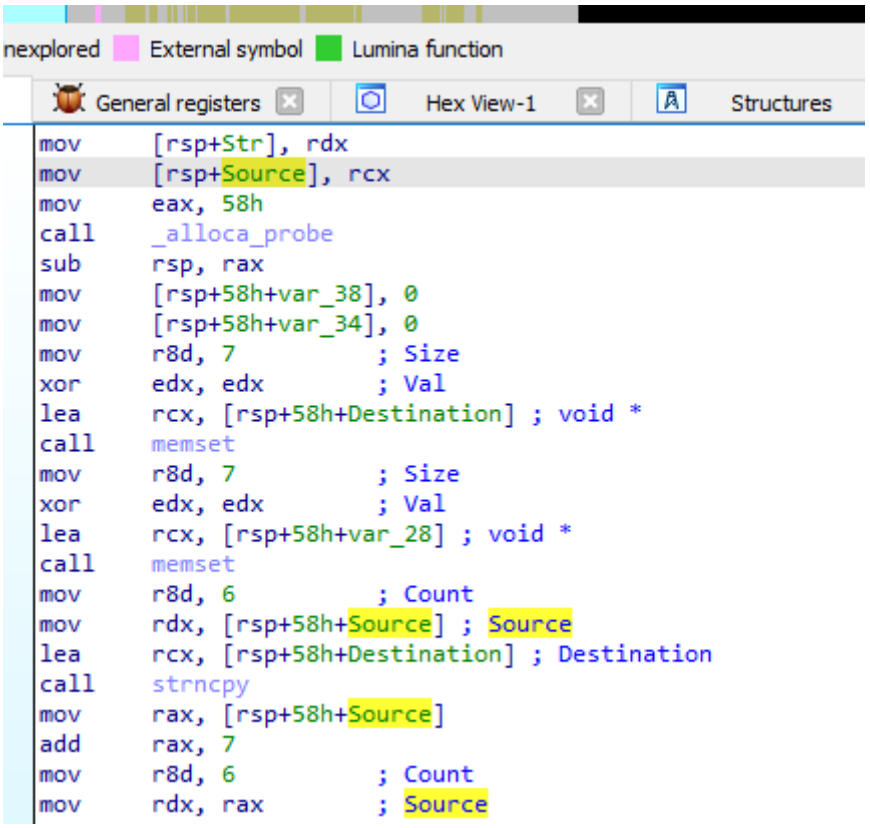
sub_140001000 переименуем magic

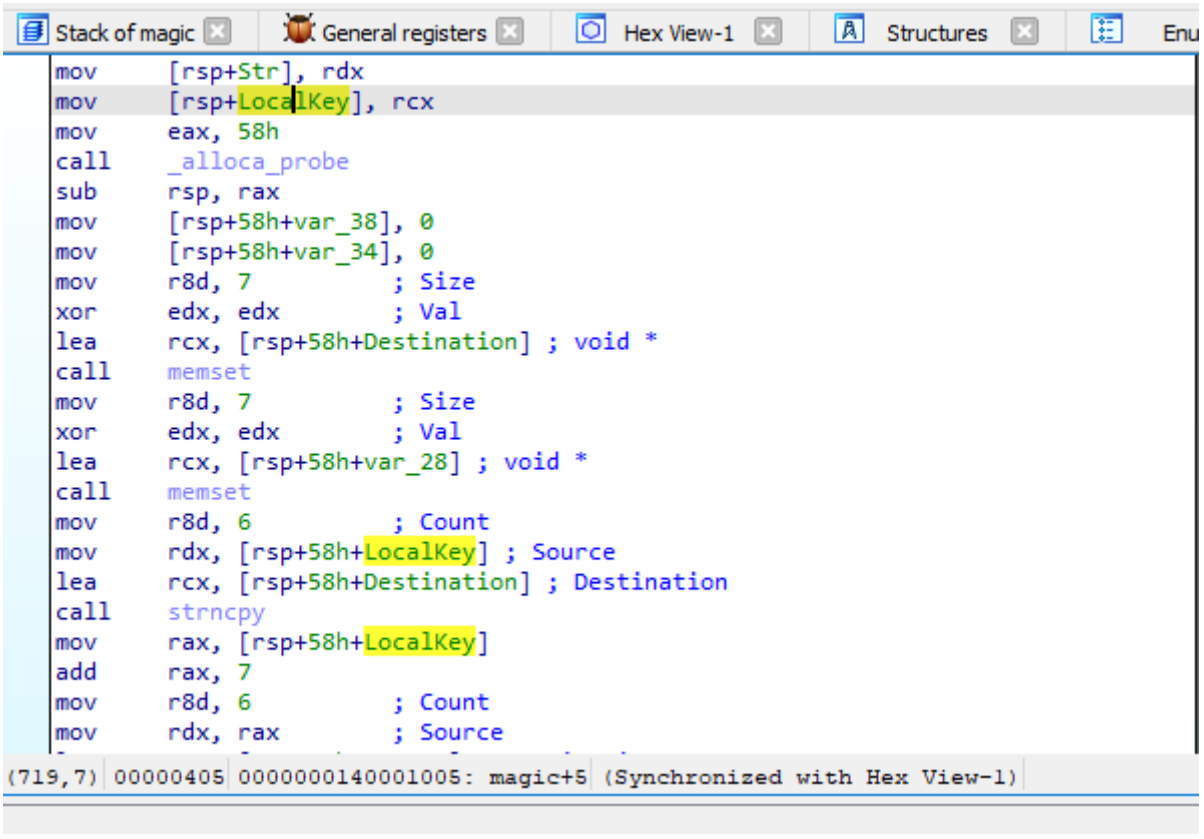
от результата данной функции зависит - правильной будет или нет

Смотрим magic

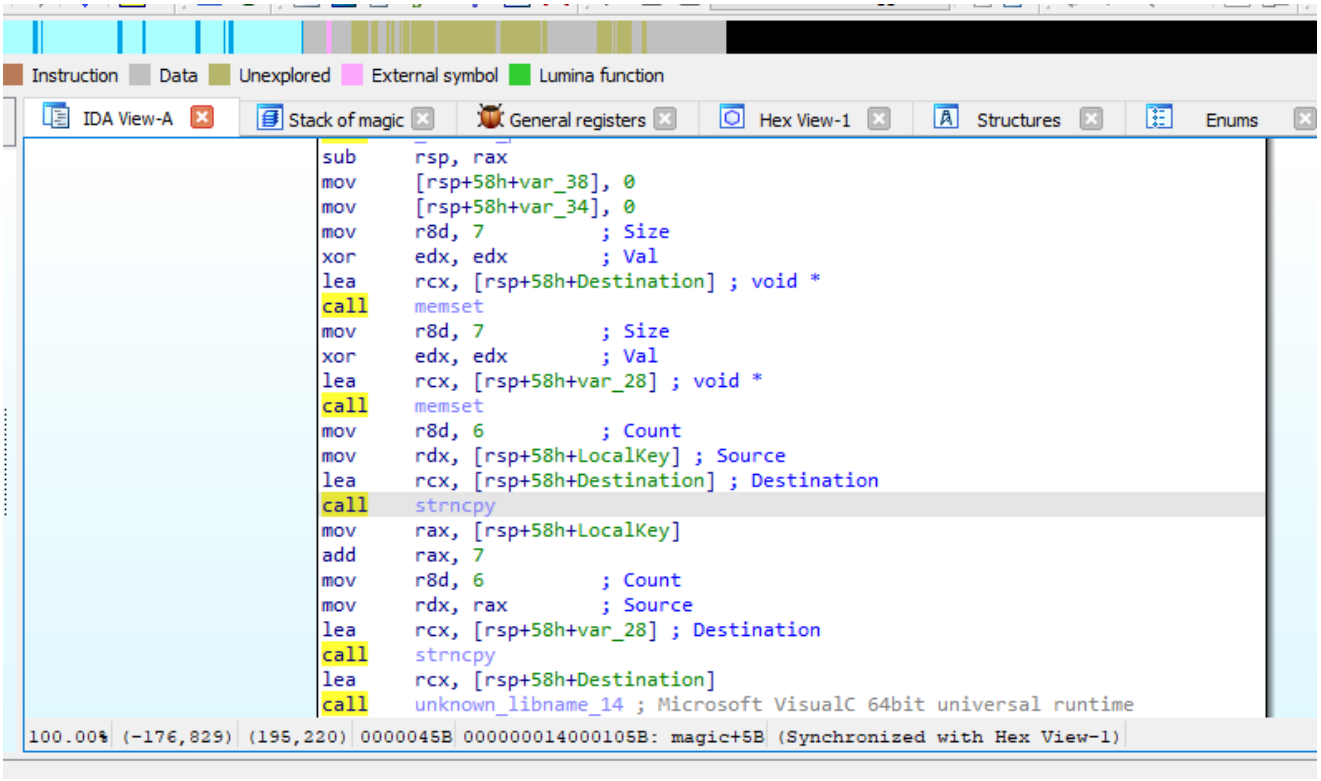




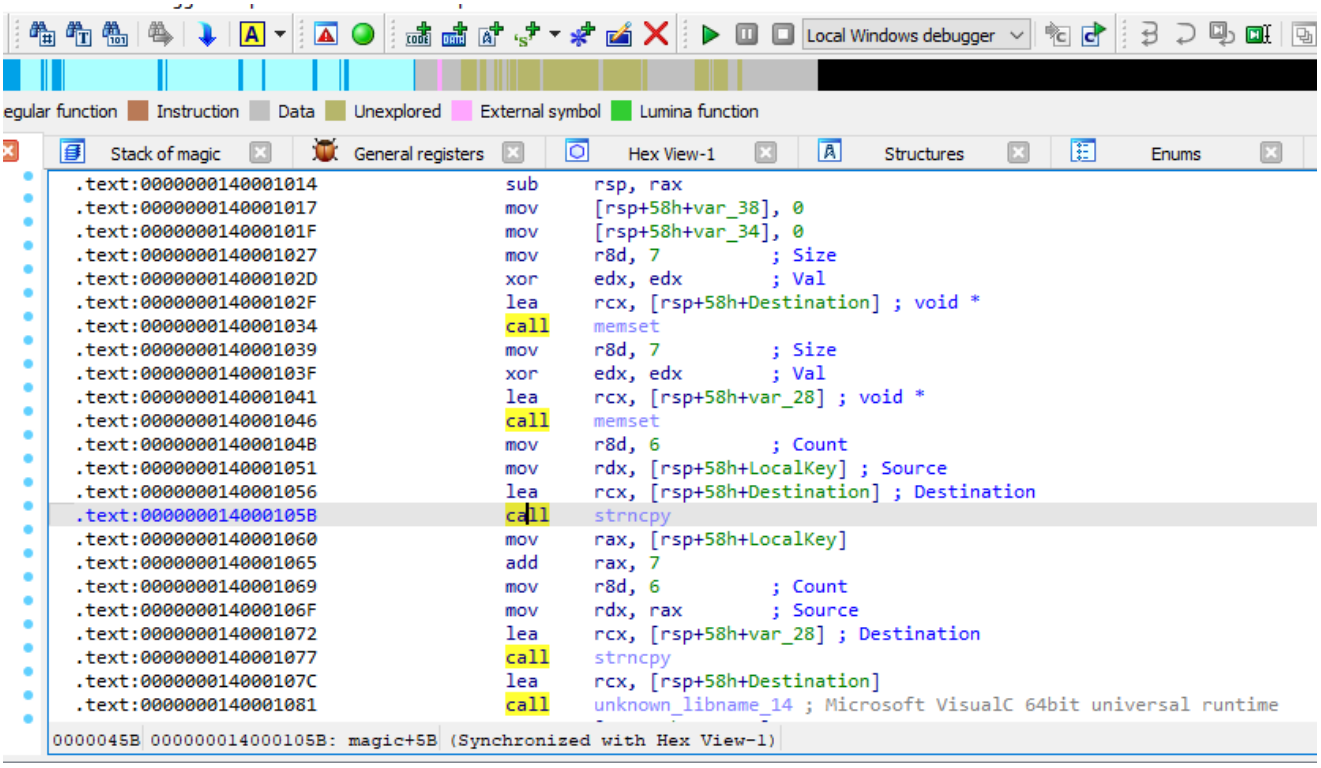

nexplored
External symbol
Lumina function
General registers
Hex View-1
Structures
mov [rsp+Str], rdx
mov [rsp+Source], rcx
mov eax, 58h
call _alloca_probe
sub rsp, rax
mov [rsp+58h+var_38], 0
mov [rsp+58h+var_34], 0
mov r8d, 7 ; Size
xor edx, edx ; Val
lea rcx, [rsp+58h+Destination] ; void *
call memset
mov r8d, 7 ; Size
xor edx, edx ; Val
lea rcx, [rsp+58h+var_28] ; void *
call memset
mov r8d, 6 ; Count
mov rdx, [rsp+58h+Source] ; Source
lea rcx, [rsp+58h+Destination] ; Destination
call strncpy
mov rax, [rsp+58h+Source]
add rax, 7
mov r8d, 6 ; Count
mov rdx, rax ; Source

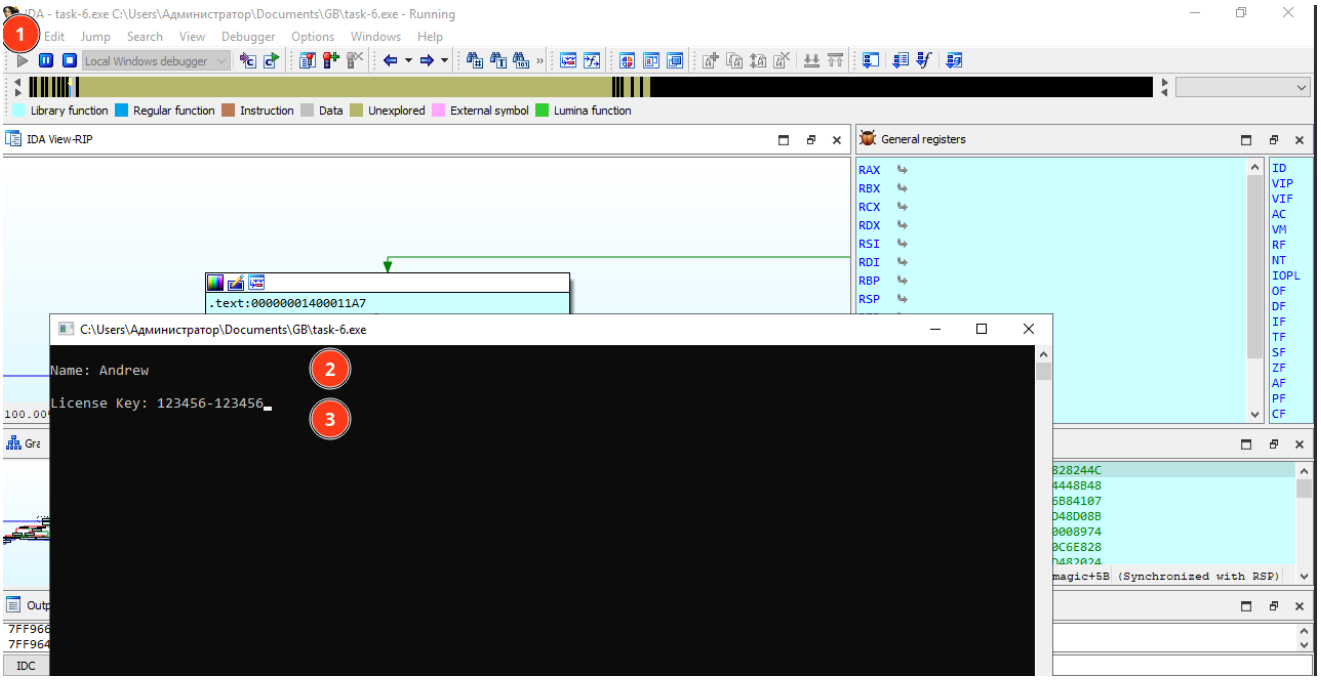

Stack of magic
General registers
Hex View-1
Structures
Enu
mov [rsp+Str], rdx
mov [rsp+LocalKey], rcx
mov eax, 58h
call _alloca_probe
sub rsp, rax
mov [rsp+58h+var_38], 0
mov [rsp+58h+var_34], 0
mov r8d, 7 ; Size
xor edx, edx ; Val
lea rcx, [rsp+58h+Destination] ; void *
call memset
mov r8d, 7 ; Size
xor edx, edx ; Val
lea rcx, [rsp+58h+var_28] ; void *
call memset
mov r8d, 6 ; Count
mov rdx, [rsp+58h+LocalKey] ; Source
lea rcx, [rsp+58h+Destination] ; Destination
call strncpy
mov rax, [rsp+58h+LocalKey]
add rax, 7
mov r8d, 6 ; Count
mov rdx, rax ; Source
(719,7) 00000405 0000000140001005: magic+5 (Synchronized with Hex View-1)

Добавим точку останова

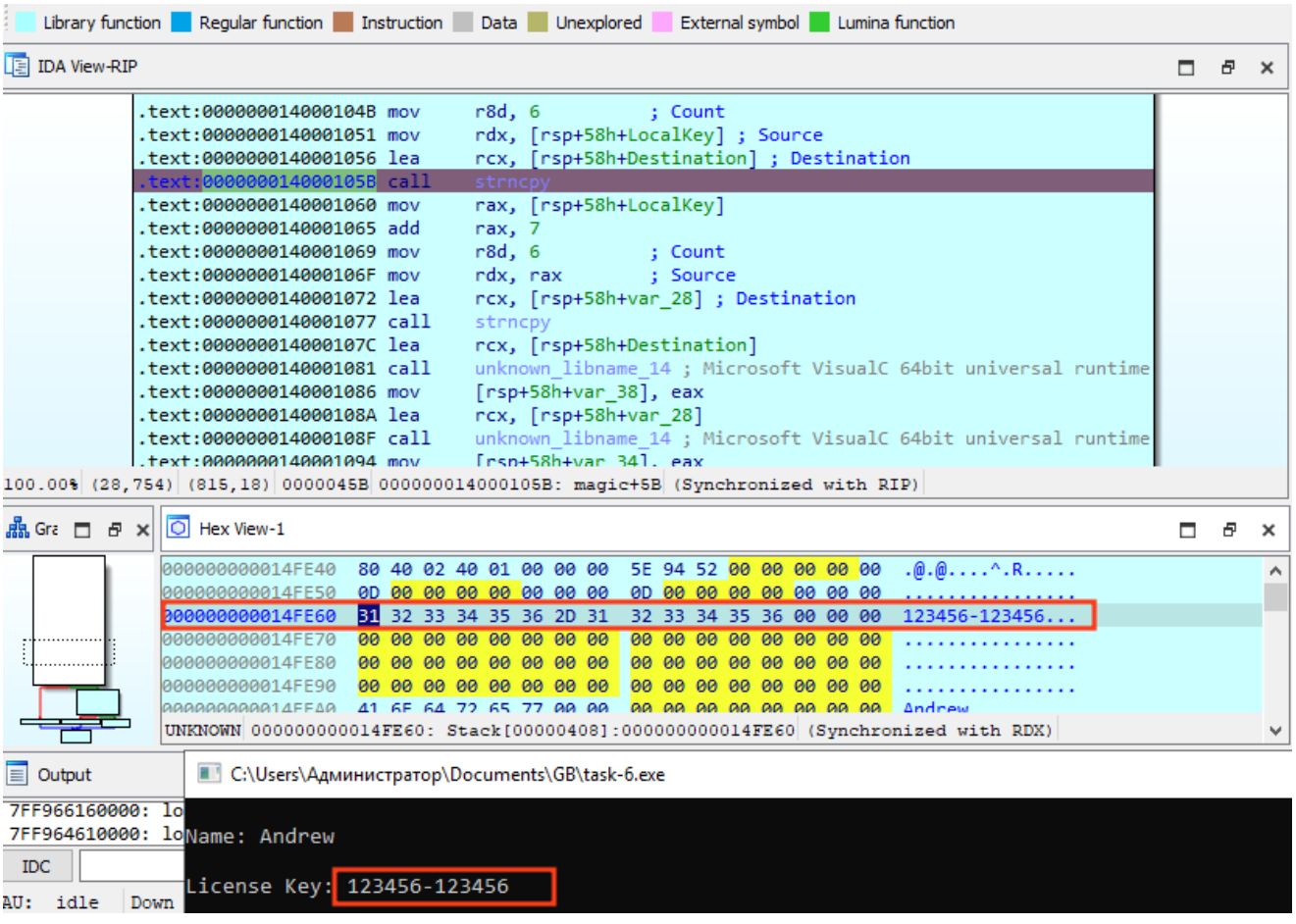


-> Text view + break point





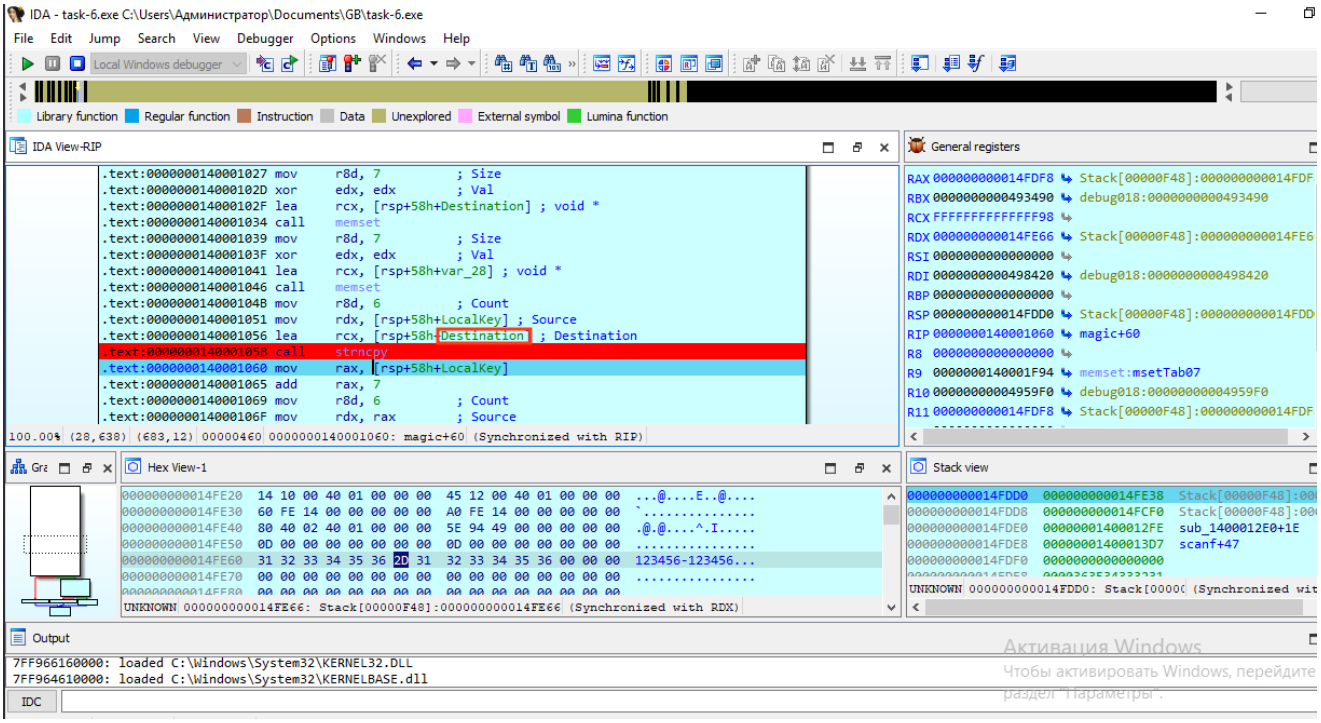
-> Synchronize with RDX



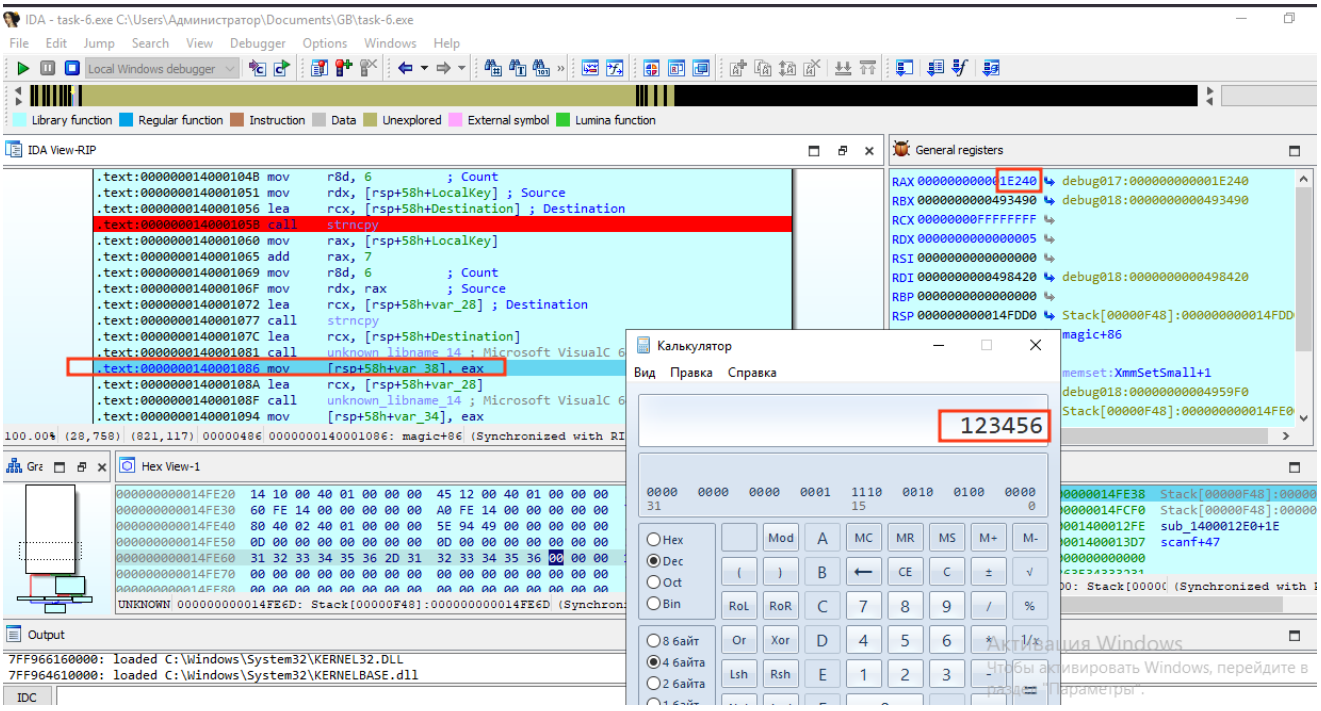
Вызываем функцию memset (F8)

Смотрим содержимое Destination = 1 2 3 4 5 6

Т.е. функция strcpy копирует байты



Смотрим далее код (F8) до eax




```
.text:0000000140001098 mov     eax, [rsp+58h+var_38]
.text:000000014000109C cdq
.text:000000014000109D mov     ecx, 0FFh
.text:00000001400010A2 idiv    ecx
.text:00000001400010A4 mov     eax, edx
.text:00000001400010A6 cdqe
.text:00000001400010A8 mov     [rsp+58h+var_20], rax
.text:00000001400010AD mov     rcx, [rsp+58h+Str] ; Str
.text:00000001400010B2 call    strlen
.text:00000001400010B7 mov     rcx, [rsp+58h+var_20]
.text:00000001400010BC cmp     rcx, rax
.text:00000001400010BF jz      short loc_1400010C5
```

idiv

происходит деление числа 123456 по модулю на 0xFF (255)

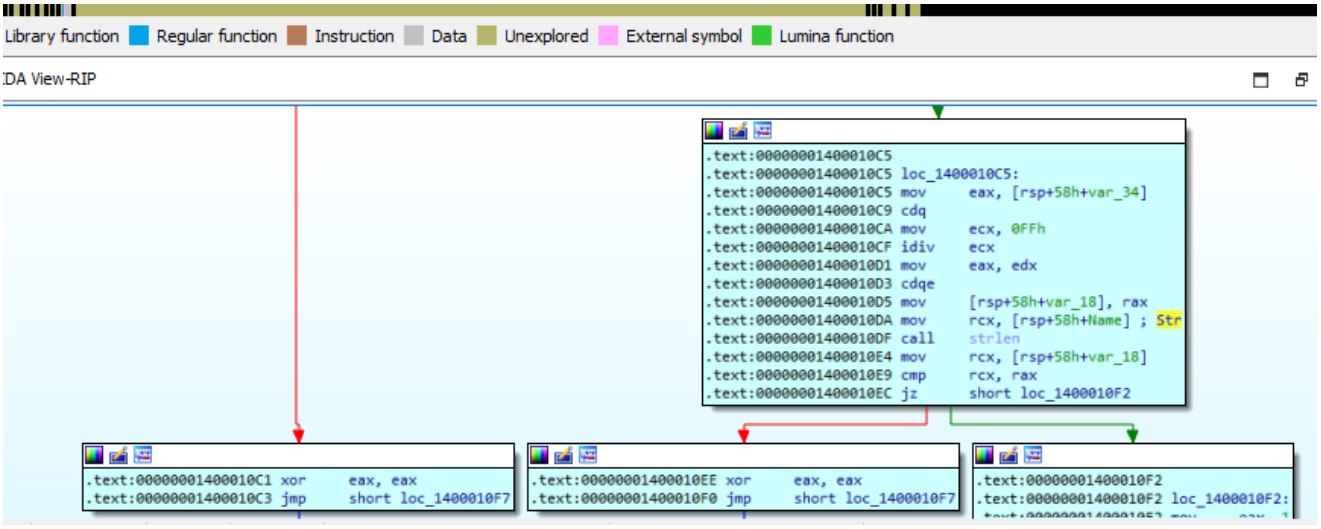
```
.text:000000014000109D mov     ecx, 0FFh
.text:00000001400010A2 idiv    ecx
.text:00000001400010A4 mov     eax, edx
.text:00000001400010A6 cdqe
.text:00000001400010A8 mov     [rsp+58h+var_20], rax
.text:00000001400010AD mov     rcx, [rsp+58h+Name] ; Str
.text:00000001400010B2 call    strlen
.text:00000001400010B7 mov     rcx, [rsp+58h+var_20]
.text:00000001400010BC cmp     rcx, rax
.text:00000001400010BF jz      short loc_1400010C5
```

Переименуем Str в Name

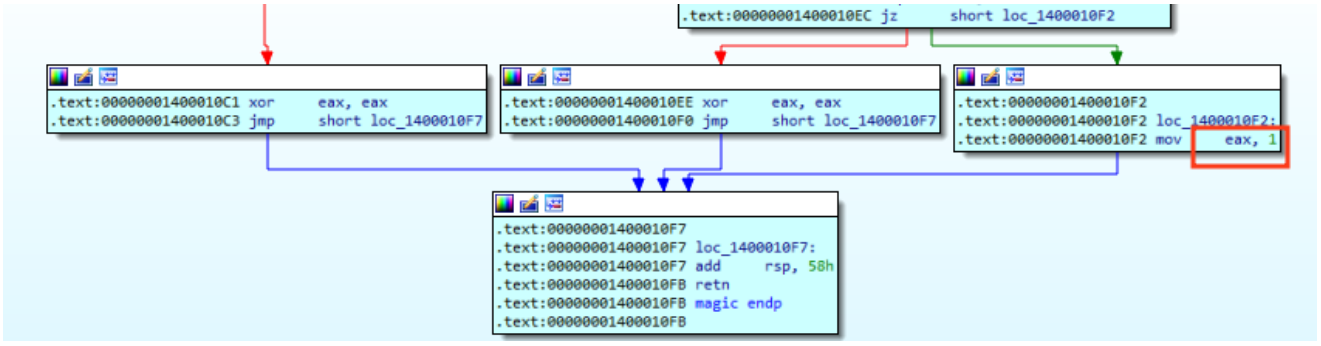
Видим, что идет сравнение в cmp длины Name и операции деления по модулю из idiv

Если значения равны, переходим к следующему блоку кода

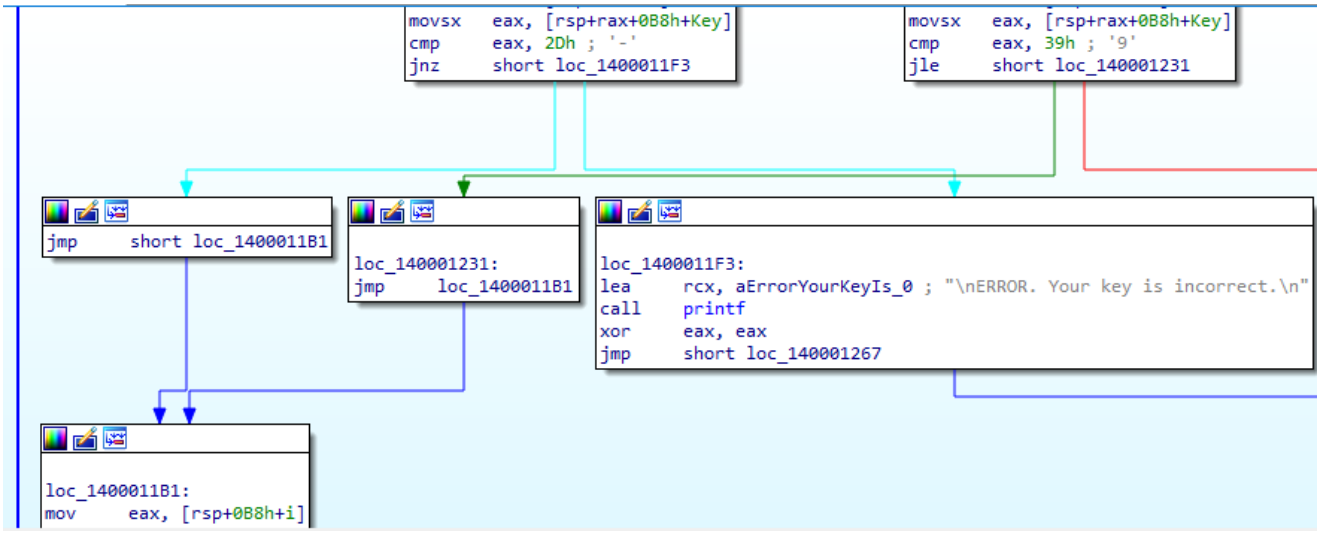
Правый блок проверить числа после тире (-123456) по аналогии с предыдущим блоком



Если сравнение успешно, в регистре eax записывается 1 и перебрасывается в блок Congratulation. Завершаем работу в данной функции.



Смотрим блок



Пробуем запустить программу:

```
C:\Users\Администратор\Documents\GB>task-6.exe

Name: Andrew

License Key: 123456-123456

ERROR. Your key is incorrect.
```

Не получилось.

Проверяем строку имя и пароль и подбираем число, проходящее все проверки

Пароль
123456-123456
0x 1E240 ...

123456 mod 255 (0x FF) = 36
Имя Andrew = длина 6
Надо получить 6
255-36 = 219
-> прибавляем к 36 число 219 (до 255) и длину имени, в нашем случае 6, т.е. плюс 225
Итак, к 123456 прибавляем число 225 и получаем число 123681

123681 mod 255 = 6, что соответствует длине имени Andrew

Проверяем

```
C:\Users\Администратор\Documents\GB>task-6.exe

Name: Andrew

License Key: 123681-123681

Congratulations! Your key is correct.
```

Использование

- Бесплатная версия IDA Freeware https://www.hex-rays.com/products/ida/support/download_freeware.shtml
- ScyllaHide https://github.com/x64dbg/ScyllaHide
- HideOD https://www.aldeid.com/wiki/OllyDbg/HideOD
- HideDebugger https://www.aldeid.com/wiki/OllyDbg/HideDebugger

Выполнил: AndreiM