13.01.2024

Курс:

Практическая работа к уроку № Lesson_10

--
Поиск уязвимостей

Задание:

Дана программа task-7. Необходимо получить ключ для вашего имени, который успешно примет программа.

Пример запуска программы:
task-7

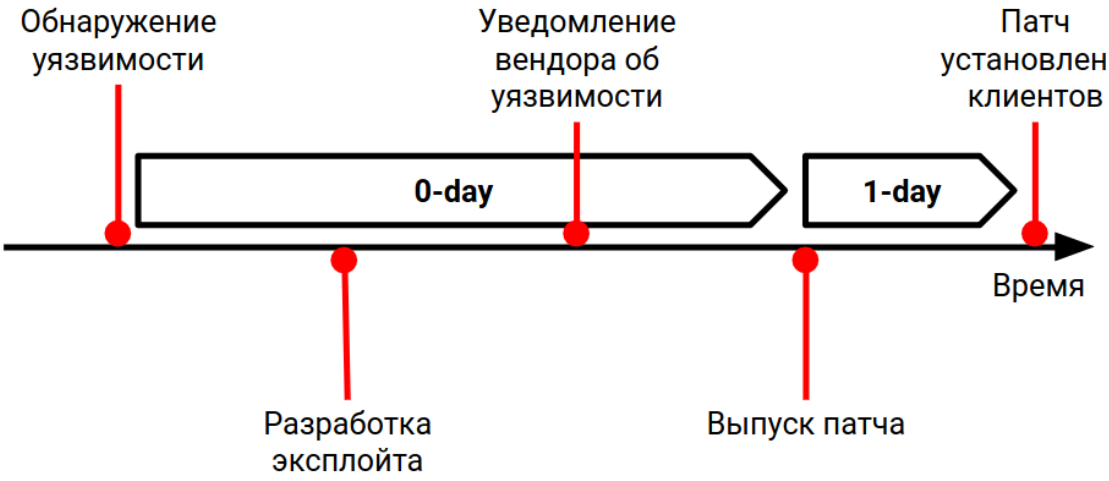
```
$ ./task-7 qweqwe
```

Примеры бинарных уязвимостей
Переполнение буфера
Целочисленные переполнения
Уязвимости формата строки

Способы поиска бинарных уязвимостей

- Фаззинг - 0-day
- Анализ исходного кода - 0-day
- Анализ дизассемблированного кода - 0-day
- Анализ патчей - 1-day

Разница между уязвимостями 0-day и 1-day



Фаззинг - техника тестирования программного обеспечения, часто автоматическая или полуавтоматическая, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных и последующий анализ результатов поведения приложения.

Анализ исходного кода может применим только тогда, когда исходный код доступен. Это не наш случай.

Анализ дизассемблированного кода - это как раз тот, чем мы занимались практически в течение всего курса. По сути это и есть реверс-инжиниринг.
Анализ патчей является очень эффективным способом для поиска 1-day уязвимостей. Поскольку, имея патч, путем сравнения двух версий бинарного файла до патча и после патча можно выявить все изменения в файле и понять, что было исправлено. Этот способ нам тоже подходит.

Состав фреймворка radare2

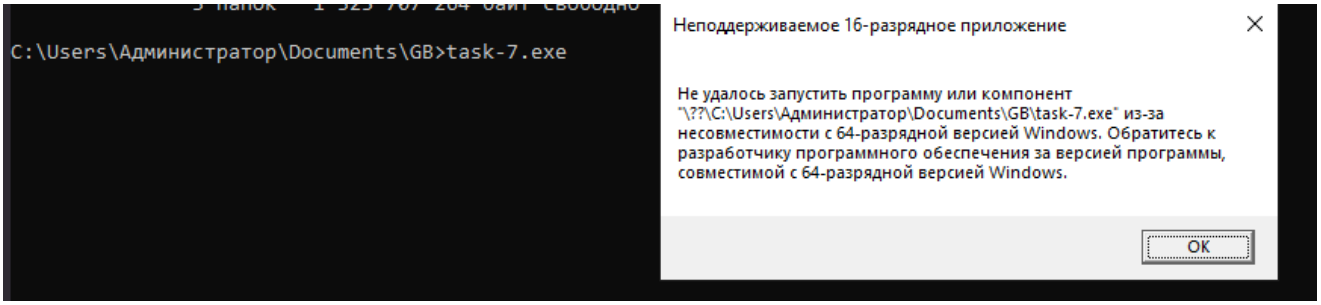
- rasm2 - дизассемблер
- rabin2
- rahash2
- radiff2
- rafind2
- ragg2
- rax2
- radeco - декомпилятор

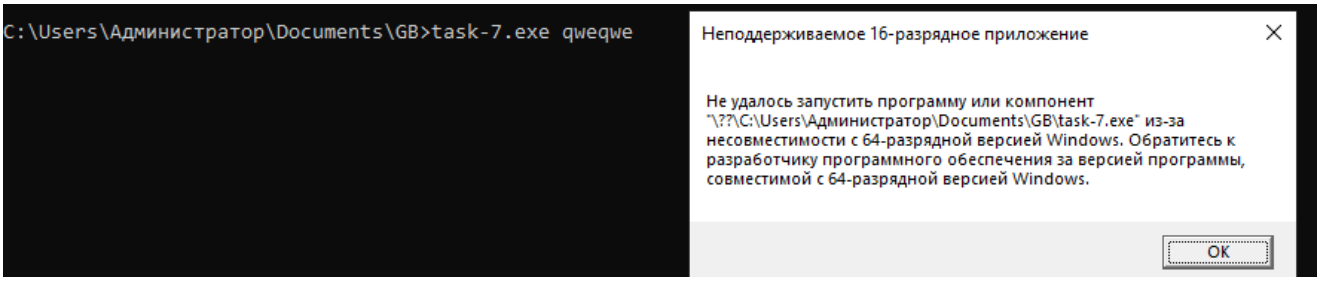
Рекомендации по установке фреймворка radare2

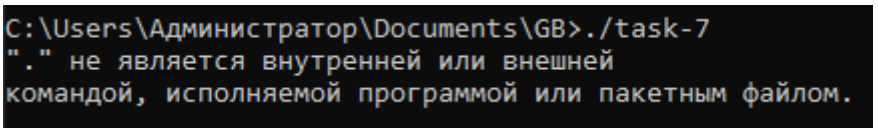
```
https://github.com/radare/radare2.git
$ git clone https://github.com/radare/radare2.git
$ cd radare2
```

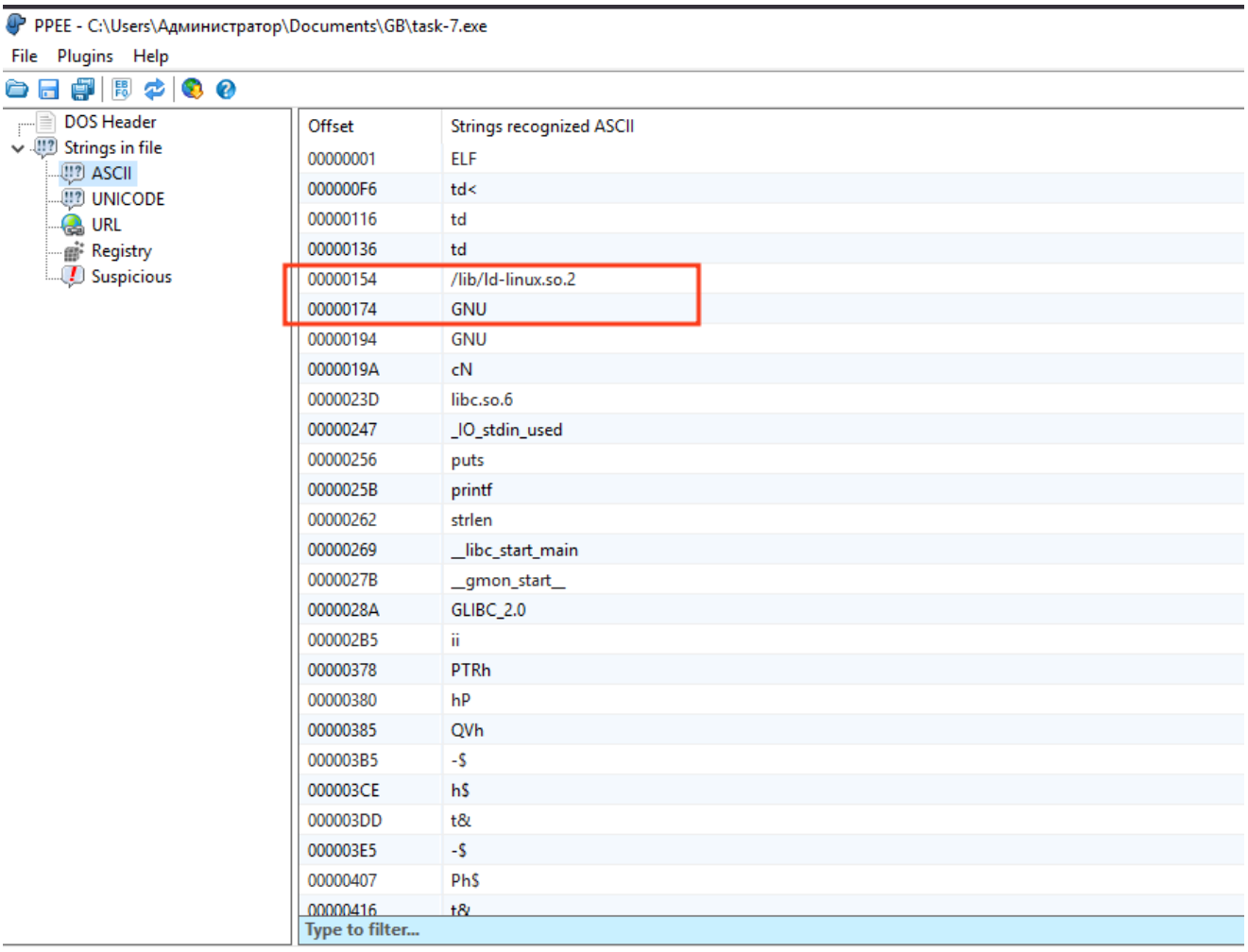
```
$ ./sys/install.sh
$ r2 -v

radare2 3.7.0-git 22267 @ linux-x86-32 git.3.6.0-99-gc1abcbe
commit: c1abcbe5e30b9050d8addbe3e9a5bb2d6fe548c3 build: 2019-07-
07__05:29:01

Обновления появляются достаточно часто, поэтому я рекомендую ежедневно их
проверять с помощью следующей команды.
$ ./sys/install.sh
```

Запускаем в CMD *task-7.exe* (Windows)



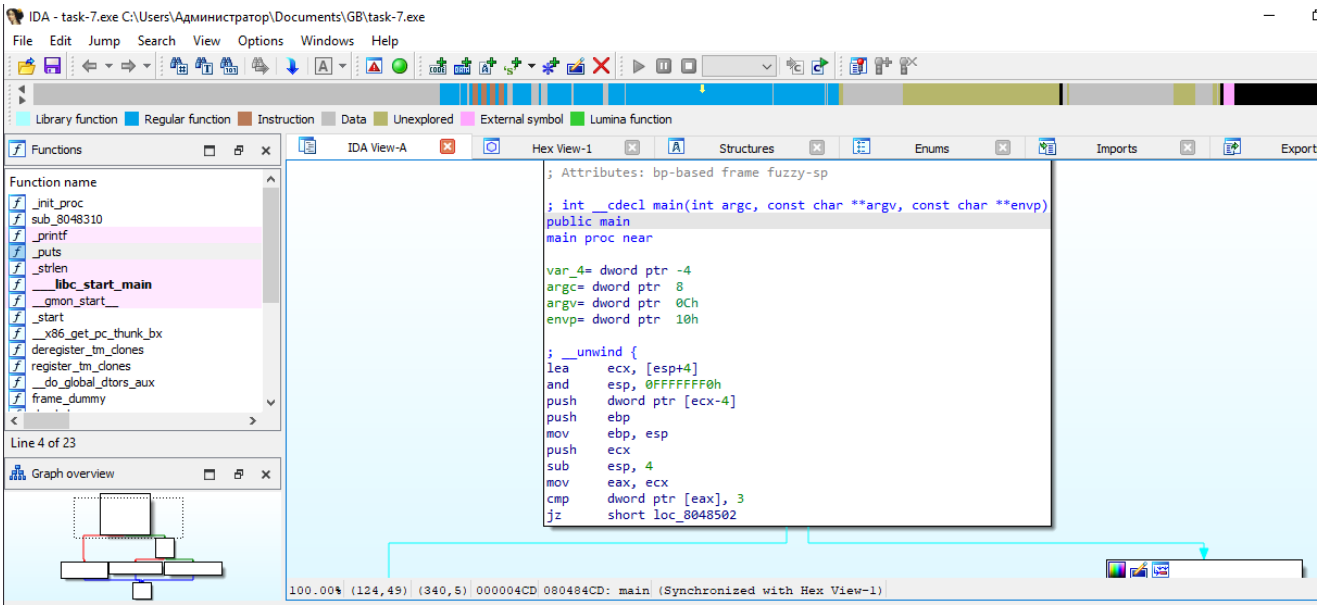


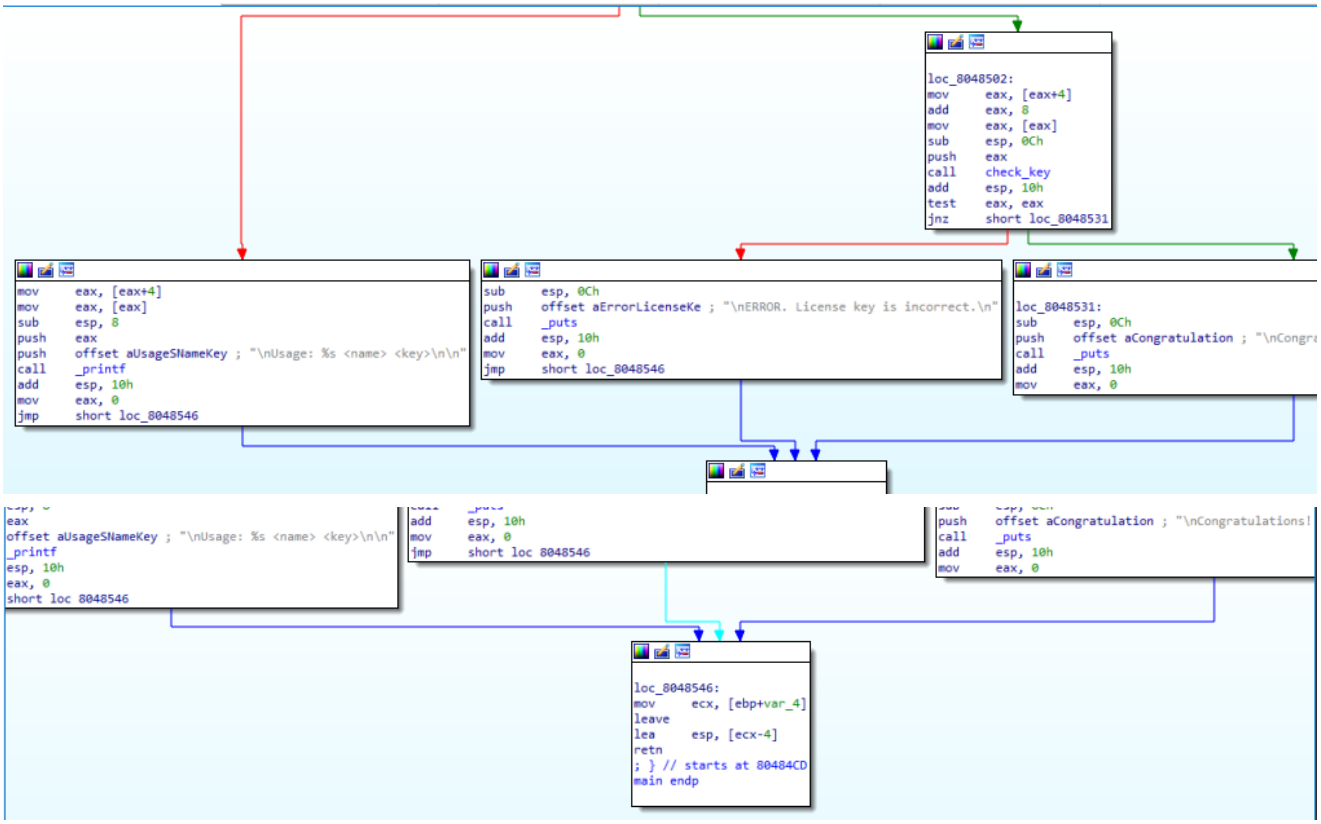




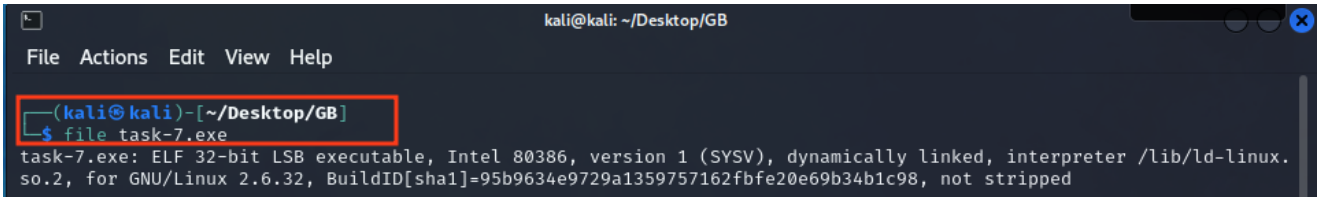
-> Linux. Видим в обозначении .ELF

IDA:





Запускаем Линукс Кали:




```
┌──(kali㉿kali)-[~/Desktop/GB]
└─$ rabin2
Usage: rabin2 [-AcdeEghHiIjlLMqrRsSUvVxzZ] [-@ at] [-a arch] [-b bits] [-B addr]
              [-C F:C:D] [-f str] [-m addr] [-n str] [-N m:M] [-P[-P] pdb]
              [-o str] [-O str] [-k query] [-D lang symname] file
```

```
┌──(kali㉿kali)-[~/Desktop/GB]
└─$ rabin2 -I task-7.exe
arch     x86
baddr    0x8048000
binsz    6210
bintype  elf
bits     32
canary   false
class    ELF32
compiler GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609
crypto   false
endian   little
havecode true
intrp    /lib/ld-linux.so.2
laddr    0x0
lang     c
```

```
linenum  true
lsyms    true
machine  Intel 80386
nx       false
os       linux
pic      false
relocs   true
relro    partial
rpath    NONE
sanitize false
static   false
stripped false
subsys   linux
va       true
```

Видим - ELF заголовок 32-битный

```
┌──(kali㉿kali)-[~/Documents]
└─$ ./task-7.exe Andrew 123456
zsh: permission denied: ./task-7.exe

┌──(kali㉿kali)-[~/Documents]
└─$ chmod +x task-7.exe

┌──(kali㉿kali)-[~/Documents]
└─$ ./task-7.exe

Usage: ./task-7.exe <name> <key>

┌──(kali㉿kali)-[~/Documents]
└─$ ./task-7.exe Andrew qweqwe
ERROR. License key is incorrect.
```

Проведем анализ программы

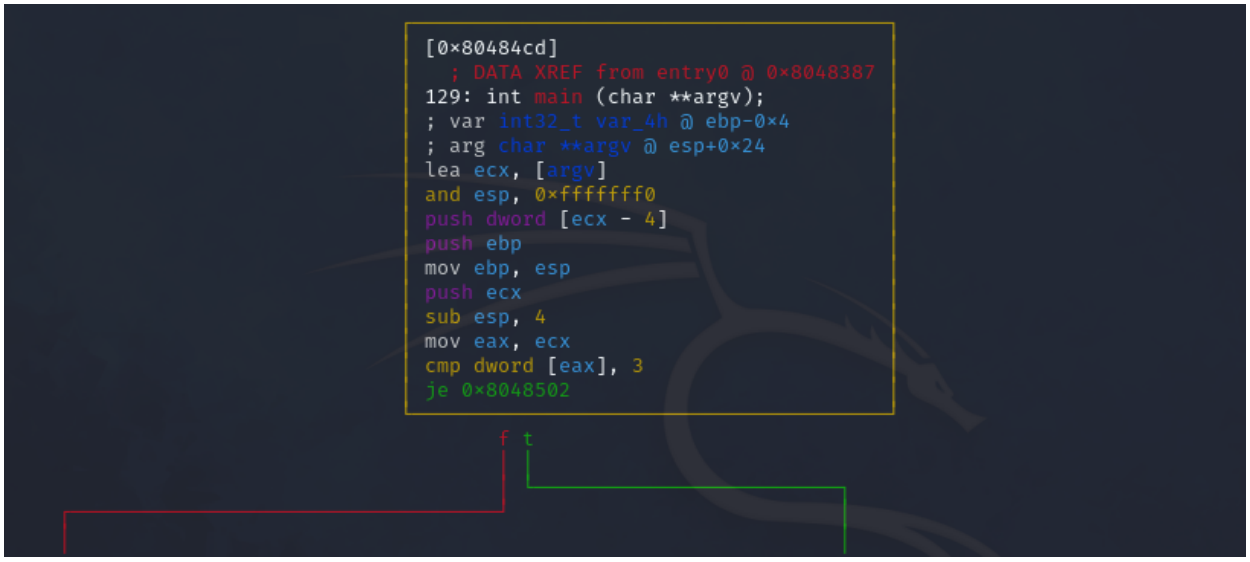
- aaa (анализ программы в r2)
- s main (главн функция)
- VV (режим графов)

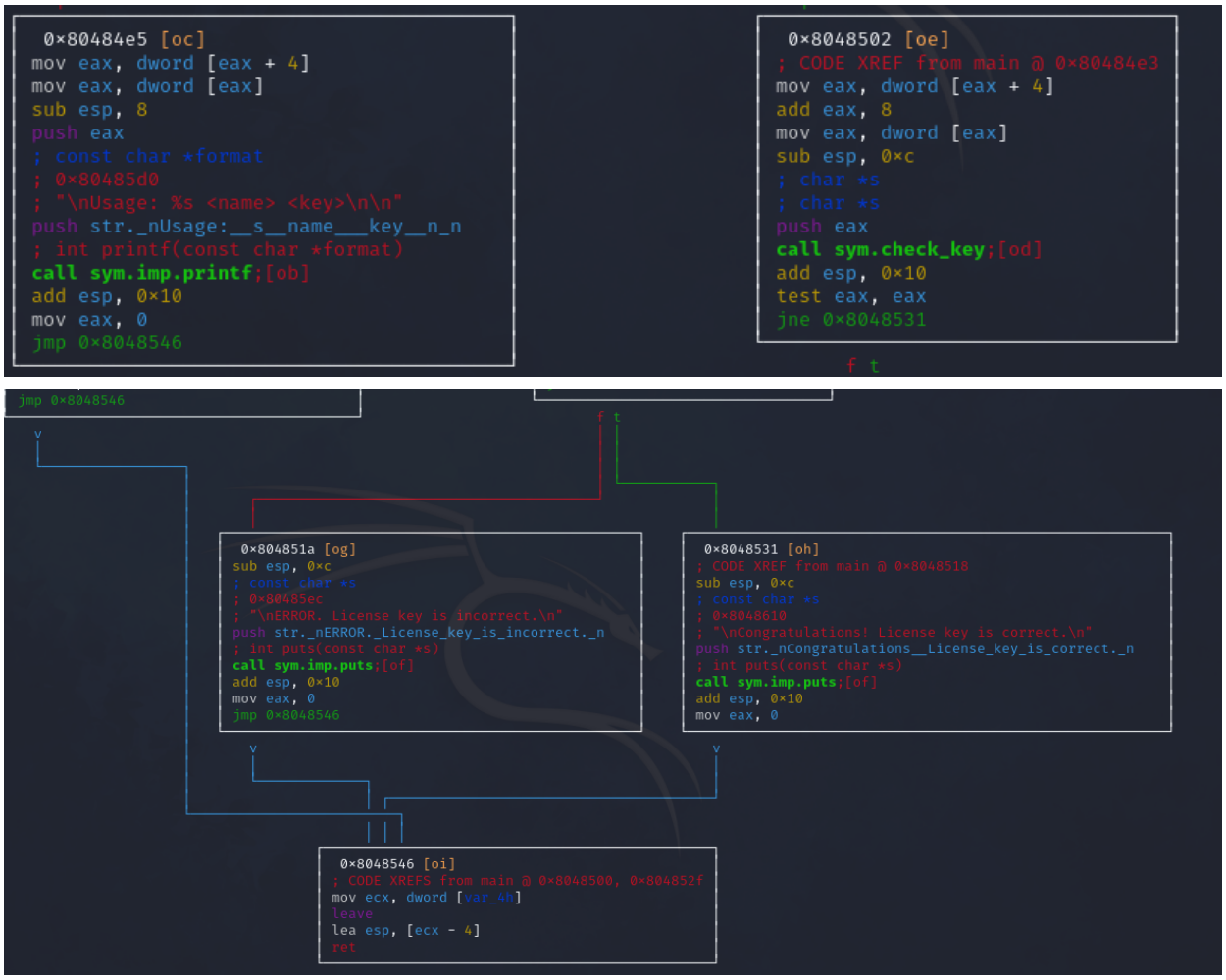
```
┌──(kali㉿kali)-[~/Documents]
└─$ r2 task-7.exe
[0x08048370]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Type matching analysis for all functions (aaft)
```

```
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x08048370]> s main
[0x080484cd]> pdf
            ; DATA XREF from entry0 @ 0x8048387
┌ 129: int main (char **argv);
│           ; var int32_t var_4h @ ebp-0x4
│           ; arg char **argv @ esp+0x24
│           0x080484cd      8d4c2404       lea ecx, [argv]
│           0x080484d1      83e4f0         and esp, 0xfffffff0
│           0x080484d4      ff71fc         push dword [ecx - 4]
│           0x080484d7      55             push ebp
│           0x080484d8      89e5           mov ebp, esp
│           0x080484da      51             push ecx
│           0x080484db      83ec04         sub esp, 4
│           0x080484de      89c8           mov eax, ecx
│           0x080484e0      833803         cmp dword [eax], 3
│       ┌─< 0x080484e3      741d           je 0x8048502
│       │   0x080484e5      8b4004         mov eax, dword [eax + 4]
│       │   0x080484e8      8b00           mov eax, dword [eax]
│       │   0x080484ea      83ec08         sub esp, 8
│       │   0x080484ed      50             push eax
│       │   0x080484ee      68d0850408     push
str._nUsage:__s__name___key__n_n ; 0x80485d0 ; "\nUsage: %s <name>
<key>\n\n" ; const char *format
│       │   0x080484f3      e828feffff     call sym.imp.printf         ;
int printf(const char *format)
│       │   0x080484f8      83c410         add esp, 0x10
│       │   0x080484fb      b800000000     mov eax, 0
│      ┌──< 0x08048500      eb44           jmp 0x8048546
│      ||   ; CODE XREF from main @ 0x80484e3
│      │└─> 0x08048502      8b4004         mov eax, dword [eax + 4]
│      │    0x08048505      83c008         add eax, 8
│      │    0x08048508      8b00           mov eax, dword [eax]
│      │    0x0804850a      83ec0c         sub esp, 0xc
│      │    0x0804850d      50             push eax                    ;
char *s
│      │    0x0804850e      e858ffffff     call sym.check_key
│      │    0x08048513      83c410         add esp, 0x10
│      │    0x08048516      85c0           test eax, eax
│      │┌─< 0x08048518      7517           jne 0x8048531
│      ||   0x0804851a      83ec0c         sub esp, 0xc
│      ||   0x0804851d      68ec850408     push
str._nERROR._License_key_is_incorrect._n ; 0x80485ec ; "\nERROR. License
key is incorrect.\n" ; const char *s
│      ||   0x08048522      e809feffff     call sym.imp.puts           ;
int puts(const char *s)
│      ||   0x08048527      83c410         add esp, 0x10
│      ||   0x0804852a      b800000000     mov eax, 0
│     ┌───< 0x0804852f      eb15           jmp 0x8048546
```

```
|     |||   ; CODE XREF from main @ 0x8048518
|     ||└─> 0x08048531      83ec0c         sub esp, 0xc
|     ||    0x08048534      6810860408     push
str._nCongratulations__License_key_is_correct._n ; 0x8048610 ;
"\nCongratulations! License key is correct.\n" ; const char *s
|     ||    0x08048539      e8f2fdffff     call sym.imp.puts           ;
int puts(const char *s)
|     ||    0x0804853e      83c410         add esp, 0x10
|     ||    0x08048541      b800000000     mov eax, 0
|     ||    ; CODE XREFS from main @ 0x8048500, 0x804852f
|     └└──> 0x08048546      8b4dfc         mov ecx, dword [var_4h]
|           0x08048549      c9             leave
|           0x0804854a      8d61fc         lea esp, [ecx - 4]
└           0x0804854d      c3             ret
```

- VV



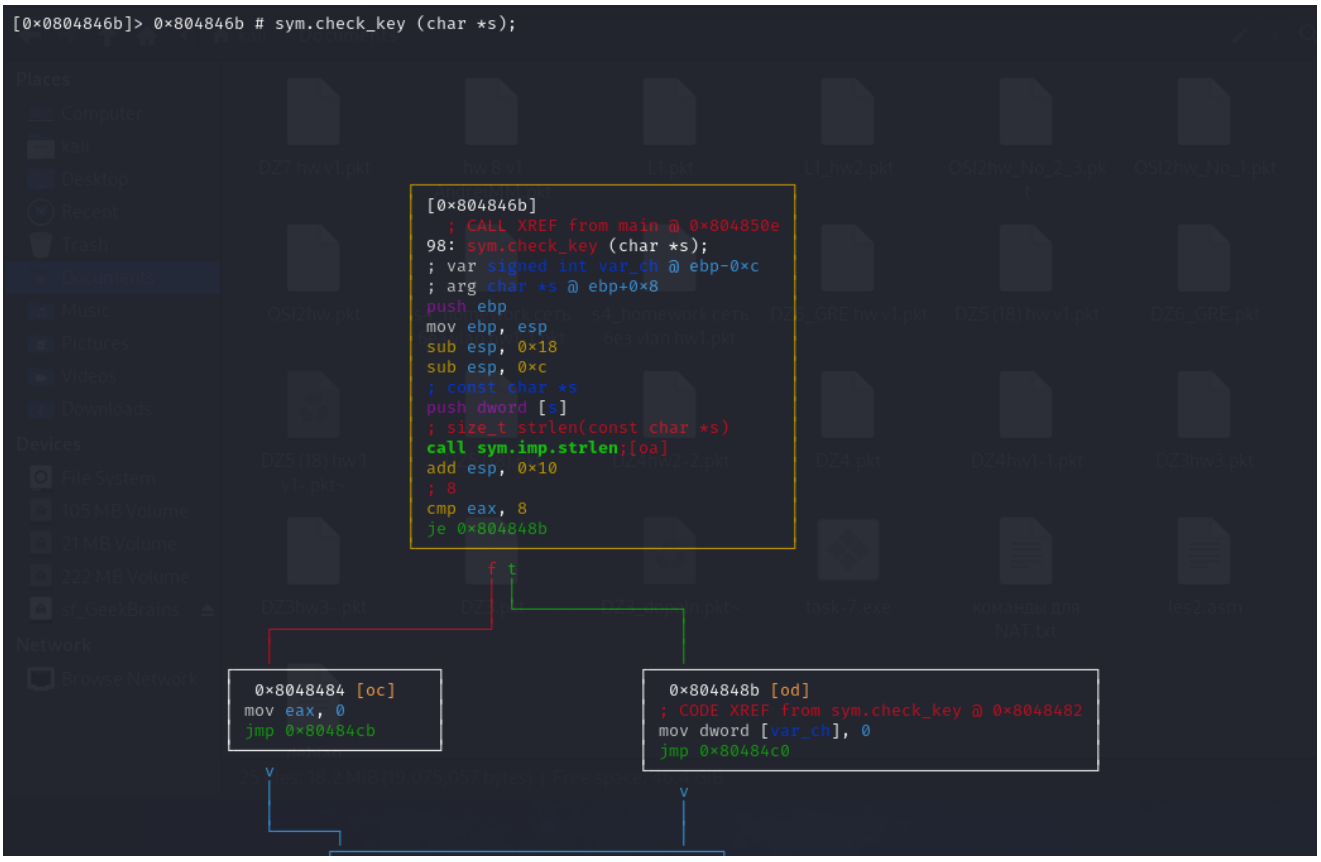


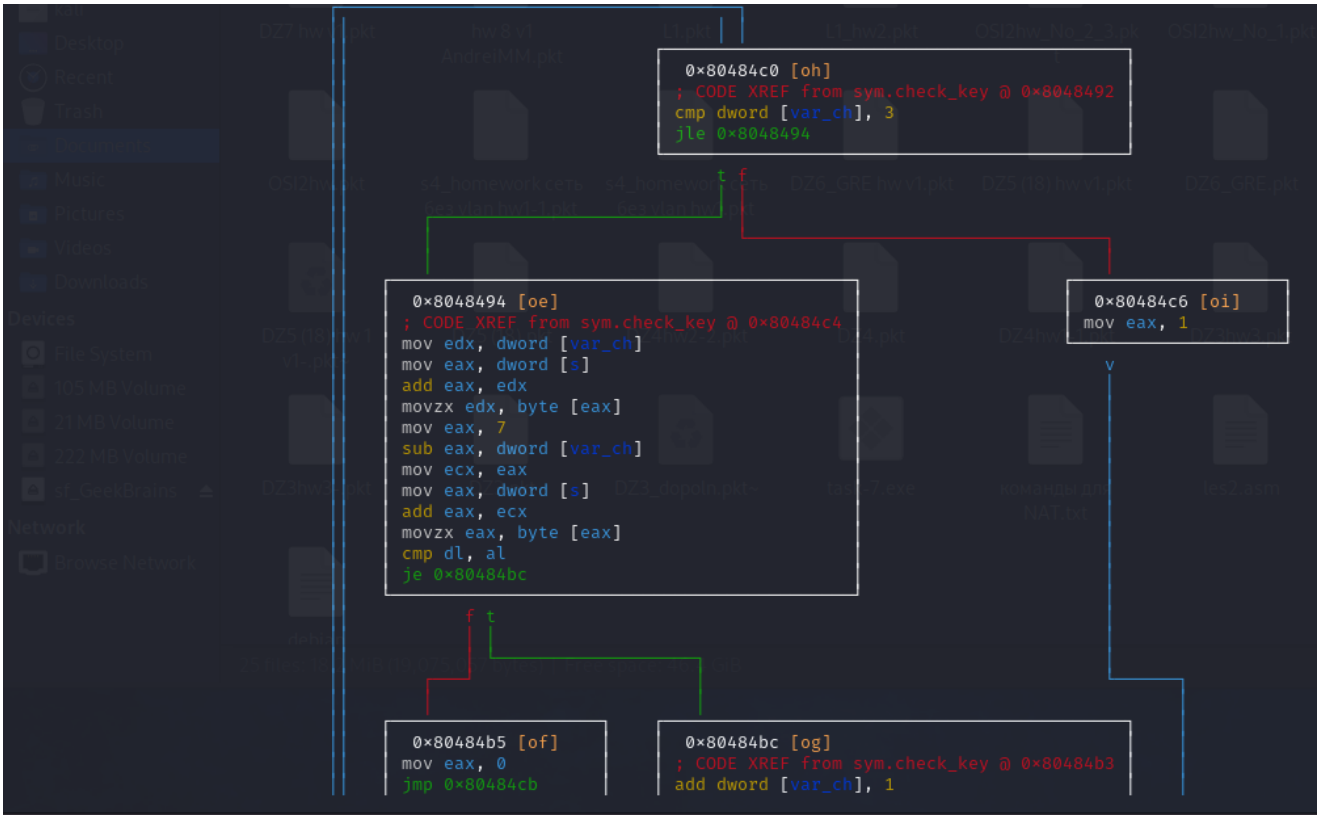
Если значение больше 0 (1), т.е. true, переходим в sym.check_key

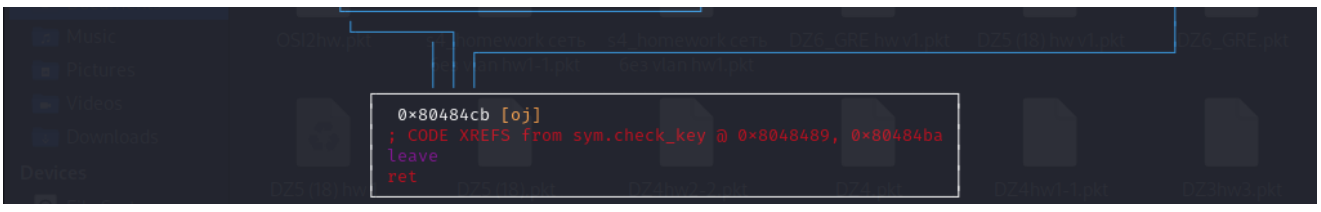
Функция sym.check_key

```
┌──(kali㉿kali)-[~/Documents]
└─$ r2 task-7.exe
[0x08048370]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x08048370]> s sym.check_key
[0x0804846b]> pdf
            ; CALL XREF from main @ 0x804850e
┌ 98: sym.check_key (char *s);
│           ; var signed int var_ch @ ebp-0xc
│           ; arg char *s @ ebp+0x8
│           0x0804846b      55              push ebp
│           0x0804846c      89e5            mov ebp, esp
│           0x0804846e      83ec18          sub esp, 0x18
│           0x08048471      83ec0c          sub esp, 0xc
```

```
|           0x08048474      ff7508         push dword [s]              ;
const char *s
|           0x08048477      e8c4feffff     call sym.imp.strlen         ;
size_t strlen(const char *s)
|           0x0804847c      83c410         add esp, 0x10
|           0x0804847f      83f808         cmp eax, 8                  ; 8
|       ┌─< 0x08048482      7407           je 0x804848b
|       │   0x08048484      b800000000     mov eax, 0
|      ┌──< 0x08048489      eb40           jmp 0x80484cb
|      ||   ; CODE XREF from sym.check_key @ 0x8048482
|      │└─> 0x0804848b      c745f4000000.  mov dword [var_ch], 0
|      │┌─< 0x08048492      eb2c           jmp 0x80484c0
|      ||   ; CODE XREF from sym.check_key @ 0x80484c4
|     ┌───> 0x08048494      8b55f4         mov edx, dword [var_ch]
|     ¦||   0x08048497      8b4508         mov eax, dword [s]
|     ¦||   0x0804849a      01d0           add eax, edx
|     ¦||   0x0804849c      0fb610         movzx edx, byte [eax]
|     ¦||   0x0804849f      b807000000     mov eax, 7
|     ¦||   0x080484a4      2b45f4         sub eax, dword [var_ch]
|     ¦||   0x080484a7      89c1           mov ecx, eax
|     ¦||   0x080484a9      8b4508         mov eax, dword [s]
|     ¦||   0x080484ac      01c8           add eax, ecx
|     ¦||   0x080484ae      0fb600         movzx eax, byte [eax]
|     ¦||   0x080484b1      38c2           cmp dl, al
|    ┌────< 0x080484b3      7407           je 0x80484bc
|    ||¦||   0x080484b5      b800000000     mov eax, 0
|   ┌─────< 0x080484ba      eb0f           jmp 0x80484cb
|   ||¦||   ; CODE XREF from sym.check_key @ 0x80484b3
|   │└────> 0x080484bc      8345f401       add dword [var_ch], 1
|   | ¦||   ; CODE XREF from sym.check_key @ 0x8048492
|   | ¦|└─> 0x080484c0      837df403       cmp dword [var_ch], 3
|   | └───< 0x080484c4      7ece           jle 0x8048494
|   |  |    0x080484c6      b801000000     mov eax, 1
|   |  |    ; CODE XREFS from sym.check_key @ 0x8048489, 0x80484ba
|   └──└──> 0x080484cb      c9             leave
└           0x080484cc      c3             ret
```

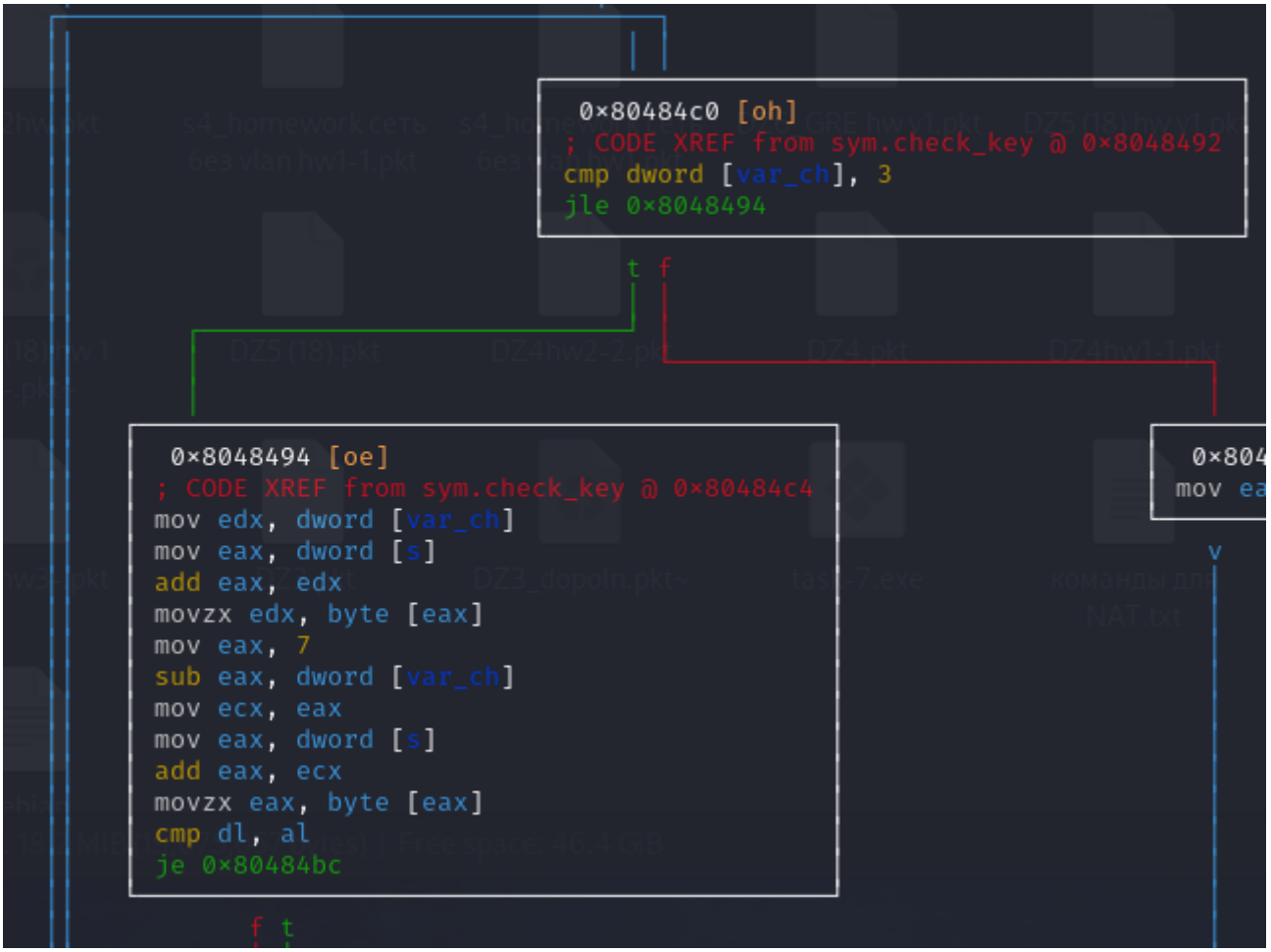

[0×0804846b]> 0×804846b # sym.check_key (char *s);
[0×804846b]
; CALL XREF from main @ 0×804850e
98: sym.check_key (char *s);
; var signed int var_ch @ ebp-0×c
; arg char *s @ ebp+0×8
push ebp
mov ebp, esp
sub esp, 0×18
sub esp, 0×c
; const char *s
push dword [s]
; size_t strlen(const char *s)
call sym.imp.strlen;[oa]
add esp, 0×10
; 8
cmp eax, 8
je 0×804848b
f t
0×8048484 [oc]
mov eax, 0
jmp 0×80484cb
0×804848b [od]
; CODE XREF from sym.check_key @ 0×8048482
mov dword [var_ch], 0
jmp 0×80484c0


0×80484c0 [oh]
; CODE XREF from sym.check_key @ 0×8048492
cmp dword [var_ch], 3
jle 0×8048494
t f
0×8048494 [oe]
; CODE XREF from sym.check_key @ 0×80484c4
mov edx, dword [var_ch]
mov eax, dword [s]
add eax, edx
movzx edx, byte [eax]
mov eax, 7
sub eax, dword [var_ch]
mov ecx, eax
mov eax, dword [s]
add eax, ecx
movzx eax, byte [eax]
cmp dl, al
je 0×80484bc
0×80484c6 [oi]
mov eax, 1
f t
0×80484b5 [of]
mov eax, 0
jmp 0×80484cb
0×80484bc [og]
; CODE XREF from sym.check_key @ 0×80484b3
add dword [var_ch], 1


0×80484cb [oj]
; CODE XREFS from sym.check_key @ 0×8048489, 0×80484ba
leave
ret

Видим в начале в коде, что сравнивается с 8

Т.е. ключ должен иметь 8 символов.



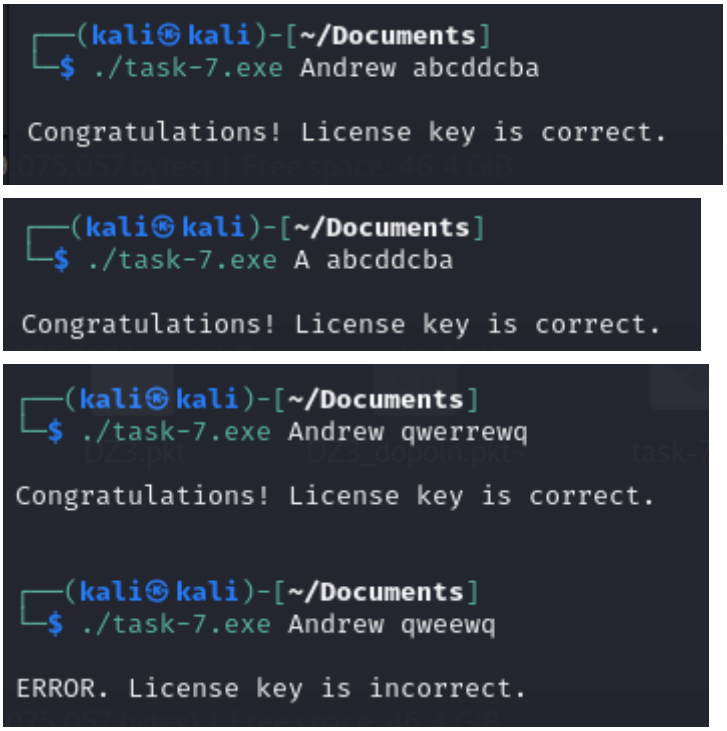
Цикл будет продолжаться до тех пор, пока var_ch будет меньше или равен 3

Далее, копируются первый байт ключа в регистр edx (movzx)
Далее добавляется в регистр eax значение 7, далее от значения 7 отнимается значение итератора (sub) равное 0, поэтому пока не изменится.
Далее указатель на последний байт ключа (add eax, ecx) и копируем его в eax (movzx).
cmp dl, al сравнение последнего и первого байта.

Видим далее после проверки, что ключ должен быть зеркальным.

Пробуем ввести ключ с 8 значениями
./task-7.exe Andrew abcddcba
./task-7.exe Andrew qwerrewq

Имя не влияет на ключ



Выполнил: AndreiM