

Открываем программу в Ghidra

Ищем определенные строки, смотрим, какие из них похожи на типичные запросы ввода пароля, статусы валидации и тому подобное

1400043d0	Enter your license key (XXXX-XXXX-XXXX):	"Enter your license key (XXXX-XXXX-XXXX-XXX...	ds
140004400	>>> License valid! Thank you for register...	">>> License valid! Thank you for regist...	ds
140004430	>>> Invalid license key. Please try again.	">>> Invalid license key. Please try again...	ds

Переходим в функцию, которая запрашивает у нас пароль

```
local_18 = DAT_140006000 ^ (ulonglong)auStack_58;
uStack_30 = 0;
local_28 = 0;
local_20 = 0xf;
key_another_copy = (void *)0x0;
FUN_140001880((basic_ostream<> *)cout_exref,"Enter your license key (XXXX-XXXX-XXXX): ");
key_input = cin_exref;
char_key_input =
    std::basic_ios<>::widen
        ((basic_ios<> *) (cin_exref + *(int *) (*(longlong *)cin_exref + 4)), '\n');
FUN_140001d40((basic_istream<> *)key_input, (longlong *)&key_another_copy, (ulonglong)char_key_input
    );
is_valid = FUN_1400011c0((longlong *)&key_another_copy);
while (is_valid == '\0') {
    FUN_140001880((basic_ostream<> *)cout_exref,">>> Invalid license key. Please try again.\n");
    FUN_140001880((basic_ostream<> *)cout_exref,"Enter your license key (XXXX-XXXX-XXXX): ");
    key_input = cin_exref;
    char_key_input =
        std::basic_ios<>::widen
            ((basic_ios<> *) (cin_exref + *(int *) (*(longlong *)cin_exref + 4)), '\n');
    FUN_140001d40((basic_istream<> *)key_input, (longlong *)&key_another_copy,
        (ulonglong)char_key_input);
    is_valid = FUN_1400011c0((longlong *)&key_another_copy);
}
FUN_140001880((basic_ostream<> *)cout_exref,">>> License valid! Thank you for registering.\n");
if (0xf < local_20) {
    _Memory = key_another_copy;
    if ((0xffff < local_20 + 1) &&
        (_Memory = *(void **) ((longlong)key_another_copy + -8),
        0x1f < (ulonglong)((longlong)key_another_copy + (-8 - (longlong)_Memory)))) {
        /* WARNING: Subroutine does not return */
        __invalid_parameter_noinfo_noreturn();
    }
    free(_Memory);
}
FUN_1400020e0(local_18 ^ (ulonglong)auStack_58);
return;
```

Судя по тому, что я смог увидеть, проверкой пароля занимается FUN_1400011c0, и статус проверки присваивает (уже переименованной мной) переменной is_valid

Посмотрим, что происходит внутри функции

```

local_10 = DAT_140006000 ^ (ulonglong)auStack_c8;
if (key[2] == 0xe) {
    start_pointer = key;
    if (0xf < (ulonglong)key[3]) {
        start_pointer = (longlong *)key;
    }
    if (*(char *)((longlong)start_pointer + 4) == '-') {
        start_pointer = key;
        if (0xf < (ulonglong)key[3]) {
            start_pointer = (longlong *)key;
        }
        if (*(char *)((longlong)start_pointer + 9) == '-') {
            local_28 = (longlong *)0x0;
            plStack_20 = (longlong *)0x0;
            local_18 = (longlong *)0x0;
            FUN_140001710(first_pack_symbol,key,0);
            FUN_140001710(second_pack_symbol,key,5);
            FUN_140001710(third_pack_symbol,key,10);
            local_28 = (longlong *)0x0;
            plStack_20 = (longlong *)0x0;
            local_18 = (longlong *)0x0;
            local_28 = (longlong *)operator_new(0x60);
            local_18 = local_28 + 0xc;
            local_98 = &local_28;
            plStack_20 = local_28;
            plStack_20 = (longlong *)
                FUN_140001ee0(first_pack_symbol, (longlong *)&local_28,local_28,&local_28);
            'eh_vector_destructor_iterator'(first_pack_symbol,0x20,3,FUN_1400016b0);
        }
    }
}

```

Сначала я увидел, что точно ключ разбивается по частям, проверяемым с помощью совпадения шаблонного символа «-»

```

FUN_140001710(first_pack_symbol,key,0);
FUN_140001710(second_pack_symbol,key,5);
FUN_140001710(third_pack_symbol,key,10);

```

Не думаю, что внутренка этой функции будет полезна. Нам нужно понять как происходит валидация ключа, а не как он распаковывается. В целом уже понятно, что 0, 5, 10 – позиции первых символов, - начала ключа, и его подключей после «-»

Честно говоря, я пытался понять, что происходит в части функции, похожей на проверку

```

if (local_28[2] == 4) {
    start_pointer = local_28;
    if (0xf < (ulonglong)local_28[3]) {
        start_pointer = (longlong *)local_28;
    }
    p1Var2 = (longlong *)((longlong)start_pointer + 4);
    for (; (start_pointer != p1Var2 &&
        (((cVar1 = (char)*start_pointer, (byte)(cVar1 - '0'U) < 10 ||
            ((byte)(cVar1 + 0xbfU) < 6)) || ((byte)(cVar1 + 0x9fU) < 6)))));
        start_pointer = (longlong *)((longlong)start_pointer + 1)) {
    }
}
start_pointer = local_28 + 4;
if (local_28[6] == 4) {
    if (0xf < (ulonglong)local_28[7]) {
        start_pointer = (longlong *)start_pointer;
    }
    p1Var2 = (longlong *)((longlong)start_pointer + 4);
    for (; (start_pointer != p1Var2 &&
        (((cVar1 = (char)*start_pointer, (byte)(cVar1 - 0x30U) < 10 ||
            ((byte)(cVar1 + 0xbfU) < 6)) || ((byte)(cVar1 + 0x9fU) < 6)))));
        start_pointer = (longlong *)((longlong)start_pointer + 1)) {
    }
}
start_pointer = local_28 + 8;
if (local_28[10] == 4) {
    if (0xf < (ulonglong)local_28[0xb]) {
        start_pointer = (longlong *)start_pointer;
    }
    p1Var2 = (longlong *)((longlong)start_pointer + 4);
    for (; (start_pointer != p1Var2 &&
        (((cVar1 = (char)*start_pointer, (byte)(cVar1 - 0x30U) < 10 ||
            ((byte)(cVar1 + 0xbfU) < 6)) || ((byte)(cVar1 + 0x9fU) < 6)))));
        start_pointer = (longlong *)((longlong)start_pointer + 1)) {
    }
}
FUN_140001a60(local_28, p1Stack_20);
start_pointer = local_28;
if ((0xffff < ((longlong)local_18 - (longlong)local_28 & 0xffffffffffffffe0U)) &&
    (start_pointer = (longlong *)local_28[-1],
    0x1f < (ulonglong)((longlong)local_28 + (-8 - (longlong)start_pointer)))) {
}

```

Но особо ничего не сообразил. Решил изучить ассемблерные инструкции. Заметил следующее

	LAB_14000144f		XREF[2]:	1400013ea(j), 14000143c(j)
14000144f 85 d2	TEST	EDX,EDX		
140001451 78 18	JS	LAB_14000146b		
140001453 85 c0	TEST	EAX,EAX		
140001455 78 14	JS	LAB_14000146b		
140001457 45 85 d2	TEST	R10D,R10D		
14000145a 78 0f	JS	LAB_14000146b		
14000145c 33 c2	XOR	EAX,EDX		
14000145e 41 03 c2	ADD	EAX,R10D		
140001461 3d ef be	CMP	EAX,0xbeef		
00 00				
140001466 0f 94 c3	SETZ	BL		
140001469 eb 02	JMP	LAB_14000146d		
	LAB_14000146b		XREF[3]:	140001451(j), 140001455(j), 14000145a(j)

Если я не ошибаюсь, то 0xBEEF – одно из «магических чисел». Встречал где-то на просторах форумов коды ошибок 0xbeef, 0xdead

Это одновременно и текст, и хекс-код – они, вроде бы, одинаковы.

Я попробовал за это уцепиться. Посмотрел в дебаггере код функций, поставил точки останова

И там тоже из всех инструкций сравнения наиболее вменяемой и похожей на валидацию была именно strcmp, BEEF

07FF70975138A	80F9 05	cmp cl,5
07FF70975138D	77 06	ja liscencevalidator.7FF7097513C5
07FF7097513BF	41:8D48 C9	lea ecx,qword ptr ds:[r8-37]
07FF7097513C3	EB 00	jmp liscencevalidator.7FF7097513D2
07FF7097513C5	41:8D48 9F	lea ecx,qword ptr ds:[r8-61]
07FF7097513C9	80F9 05	cmp cl,5
07FF7097513CC	77 10	ja liscencevalidator.7FF7097513DE
07FF7097513CE	41:8D48 A9	lea ecx,qword ptr ds:[r8-57]
07FF7097513D2	08C1	or eax,ecx
07FF7097513D4	49:FFC1	inc r9
07FF7097513D7	40:38CB	cmp r9,r11
07FF7097513DA	75 C4	jne liscencevalidator.7FF7097513A0
07FF7097513DC	EB 03	jmp liscencevalidator.7FF7097513E1
07FF7097513DE	41:8BC2	mov eax,r10d
07FF7097513E1	48:8D4F 40	lea rcx,qword ptr ds:[rdi+40]
07FF7097513E5	48:8379 10 04	cmp qword ptr ds:[rcx+10],4
07FF7097513EA	75 63	jne liscencevalidator.7FF70975144F
07FF7097513EC	48:8379 18 0F	cmp qword ptr ds:[rcx+18],F
07FF7097513F1	76 0B	jbe liscencevalidator.7FF7097513FE
07FF7097513F3	4C:8B09	mov r9,qword ptr ds:[rcx]
07FF7097513F6	4D:8BD9	mov r11,r9
07FF7097513F9	49:8BC9	mov rcx,r9
07FF7097513FC	EB 06	jmp liscencevalidator.7FF709751404
07FF7097513FE	4C:8BC9	mov r9,rcx
07FF709751401	4C:8BD9	mov r11,rcx
07FF709751404	49:83C3 04	add r11,4
07FF709751408	49:38CB	cmp rcx,r11
07FF70975140B	74 3F	je liscencevalidator.7FF70975144C
07FF70975140D	0F1F00	nop dword ptr ds:[rax],eax
07FF709751410	45:0FBE01	movsx r8d,byte ptr ds:[r9]
07FF709751414	C1E3 04	shl ebx,4
07FF709751417	41:8D48 D0	lea ecx,qword ptr ds:[r8-30]
07FF70975141B	80F9 09	cmp cl,9
07FF70975141E	77 06	ja liscencevalidator.7FF709751426
07FF709751420	41:8D48 D0	lea ecx,qword ptr ds:[r8-30]
07FF709751424	EB 1C	jmp liscencevalidator.7FF709751442
07FF709751426	41:8D48 BF	lea ecx,qword ptr ds:[r8-41]
07FF70975142A	80F9 05	cmp cl,5
07FF70975142D	77 06	ja liscencevalidator.7FF709751435
07FF70975142F	41:8D48 C9	lea ecx,qword ptr ds:[r8-37]
07FF709751433	EB 00	jmp liscencevalidator.7FF709751442
07FF709751435	41:8D48 9F	lea ecx,qword ptr ds:[r8-61]
07FF709751439	80F9 05	cmp cl,5
07FF70975143C	77 11	ja liscencevalidator.7FF70975144F
07FF70975143E	41:8D48 A9	lea ecx,qword ptr ds:[r8-57]
07FF709751442	08D9	or ebx,ecx
07FF709751444	49:FFC1	inc r9
07FF709751447	40:38CB	cmp r9,r11
07FF70975144A	75 C4	jne liscencevalidator.7FF709751410
07FF70975144C	44:8BD3	mov r10d,ebx
07FF70975144F	85D2	test edx,edx
07FF709751451	78 18	js liscencevalidator.7FF709751468
07FF709751453	85C0	test eax,eax
07FF709751455	78 14	js liscencevalidator.7FF709751468
07FF709751457	45:85D2	test r10d,r10d
07FF70975145A	78 0F	js liscencevalidator.7FF709751468
07FF70975145C	33C2	xor eax,edx
07FF70975145E	41:03C2	add eax,r10d
07FF709751461	3D EFBE0000	cmp eax,BEEF
07FF709751466	0F94C3	sete bl
07FF709751469	EB 02	jmp liscencevalidator.7FF70975146D
07FF70975146B	32DB	xor bl,bl
07FF70975146D	48:8B9424 A8000000	mov rdx,qword ptr ss:[rsp+A8]
07FF709751475	48:8BCF	mov rcx,rdi
07FF709751478	E8 E3050000	call liscencevalidator.7FF709751A60
07FF70975147D	48:8B9424 B0000000	mov rdx,qword ptr ss:[rsp+B0]
07FF709751485	48:8B8C24 A0000000	mov rcx,qword ptr ss:[rsp+A0]
07FF70975148D	48:2BD1	sub rdx,rcx
07FF709751490	48:83E2 E0	and rdx,FFFFFFFFFFFFFFE0
07FF709751494	48:8BC1	mov rax,rcx
07FF709751497	48:81FA 00100000	cmp rdx,1000
07FF70975149E	72 1C	jb liscencevalidator.7FF70975148C
07FF7097514A0	48:83C2 27	add rdx,27
07FF7097514A4	48:8B49 F8	mov rcx,qword ptr ds:[rcx-8]
07FF7097514A8	48:2BC1	sub rax,rcx
07FF7097514AB	48:83C0 F8	add rax,FFFFFFFFFFFFFFF8
07FF7097514AF	48:83F8 1F	cmp rax,1F
07FF7097514B3	76 07	jbe liscencevalidator.7FF70975148C

Я решил посмотреть (впервые после начала изучения реверса), как выглядит функция валидации полностью в качестве графа

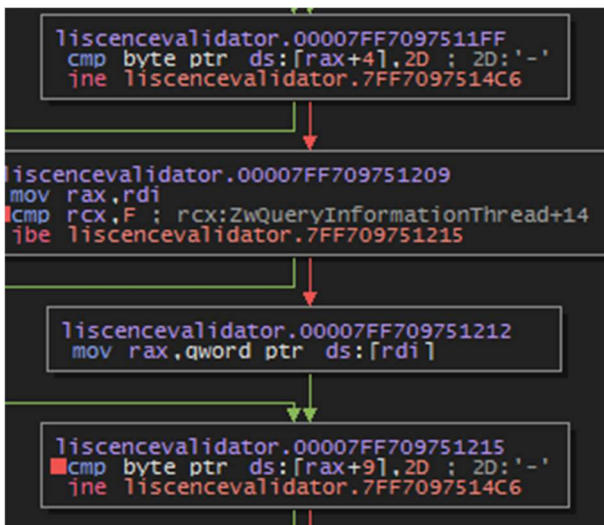
И начал сопоставлять вызовы ветвлений с дизассемблированным C-подобным кодом

```

xor ebx,ebx
cmp qword ptr ds:[rcx+10],E ; rcx+10:NtOpenProcess+4
jne liscencevalidator.7FF7097514C6

```

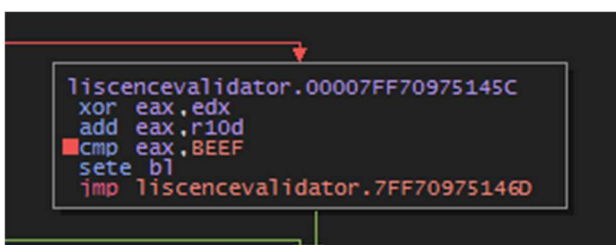
```
if (key[2] == 0xe) {
    start_pointer = key;
```



```
if (*(char *)((longlong)start_pointer + 4) == '-') {
    start_pointer = key;
    if (0xf < (ulonglong)key[3]) {
        start_pointer = (longlong *)key;
    }
    if (*(char *)((longlong)start_pointer + 9) == '-') {
```

Не скажу, что мне это много дало – видимо, не хватает каких-то знаний

Но снова заметил



Ну раз я уже несколько раз в важных местах встречаю это, то попробую самый тупой способ – просто введу ключ с таким содержимым

```
Enter your license key (XXXX-XXXX-XXXX): BEEF-BEEF-BEEF
>>> License valid! Thank you for registering.
```

К счастью, все получилось.

После такого удачного стечения обстоятельств я полез читать комментарии других выполнивших задание и загруженные райтапы

Несколько раз встретил комментарии, в которых говорилось, что в IDA Pro показывается следующая интерпретация валидации ключа

```
```cpp
v27 = v8 >= 0 && v15 >= 0 && v7 >= 0 && v7 + (v8 ^ v15) == 0xBEEF;
```
```

Значит, можно было бы разобрать её. Вполне возможно, стоит озаботиться переходом либо на IDA Free, либо поиском какого-нибудь способа получить IDA Pro. Но это, наверное, сильно позже

Попробуем работать с тем, что есть

Значит, у нас есть три подключа – v8, v7, v15. Каждый из них не равен нулю, а $v7 + (v8 \text{ xor } v15)$ должны дать 0xBEEF

Тогда пойдем от обратного. Пусть $v7 = \text{AAAA}$

Тогда $v8 \text{ xor } v15$ будет должно быть 0x1445. Ну пусть в таком случае v8 будет 0x1445, $v15 = 0x0000$. Попробуем

```
Enter your license key (XXXX-XXXX-XXXX): AAAA-1445-0000  
>>> License valid! Thank you for registering.
```

Работает!

Тогда понятно, почему ключ BEEF-BEEF-BEEF подошел
xor одинаковых значений дал 0, и в результате $\text{BEEF} == \text{BEEF}$, что истинно.