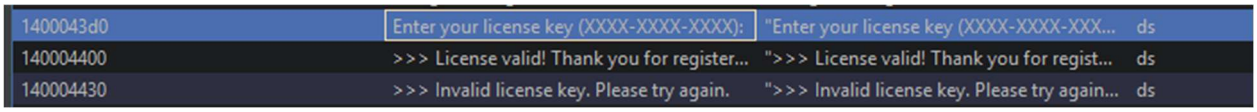
Открываем программу в Ghidra

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



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_14000ld40((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_14000ld40((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 ((0xfff < 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 внутрянка 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;
  }
  plVar2 = (longlong *)((longlong)start_pointer + 4);
  for (; (start_pointer != plVar2 &&
         (((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;
  }
  plVar2 = (longlong *)((longlong)start_pointer + 4);
  for (; (start_pointer != plVar2 &&
         (((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;
  }
  plVar2 = (longlong *)((longlong)start_pointer + 4);
  for (; (start_pointer != plVar2 &&
         (((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,plStack_20);
start_pointer = local_28;
if ((0xfff < ((longlong)local_18 - (longlong)local_28 & 0xfffffffffffffe0U)) &&
   (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

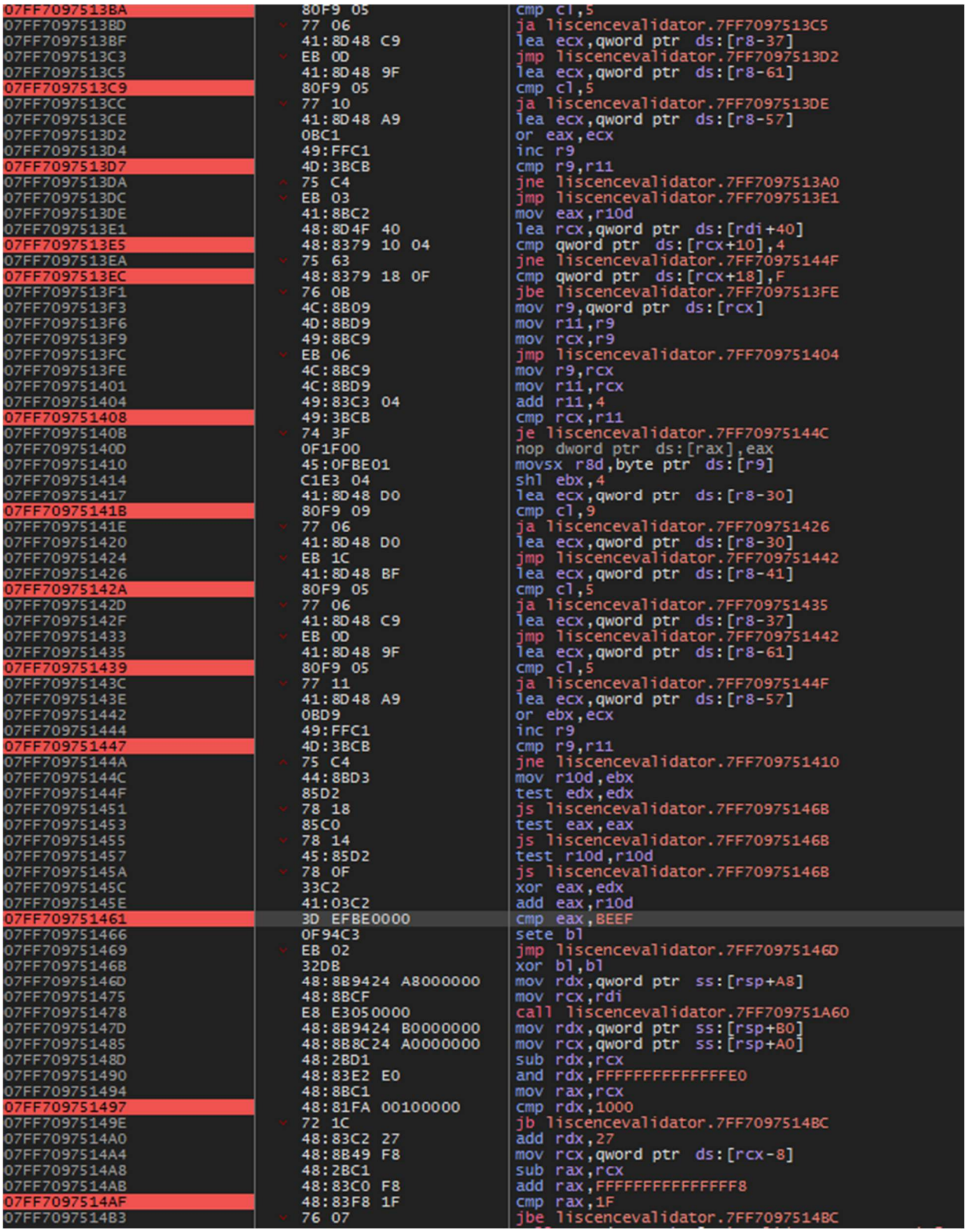
                    LAB_14000146b                                   XREF[3]:     140001451(j), 140001455(j),
```

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.

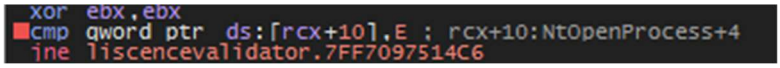
Я попробовал за это уцепиться. Посмотрел в дебаггере код функций, поставил точки останова

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

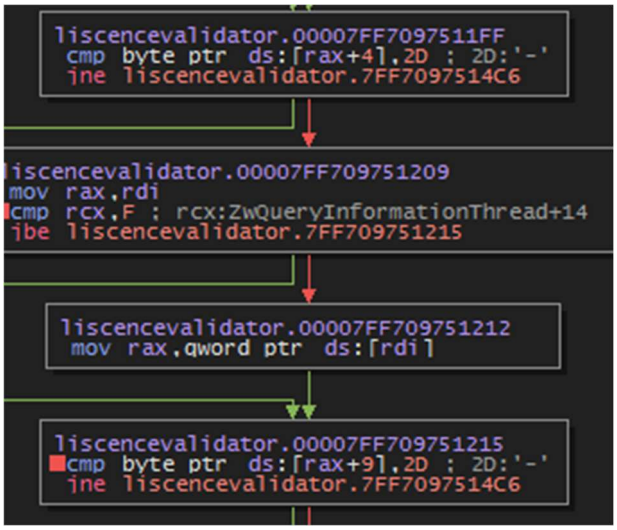


Я решил посмотреть (впервые после начала изучения реверса), как выглядит функция валидации полностью в качестве графа

И начал сопоставлять вызовы ветвлений с дизассмеблированным C-подобным кодом



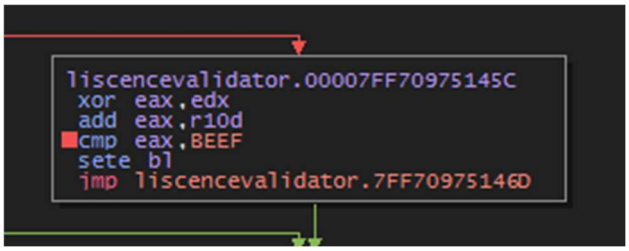
```
if (key[2] == 0xe) {
  start_pointer = key;
```




```
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 xor v15) должны дать 0xBEEF

Тогда пойдем от обратного. Пусть v7 = AAAA

Тогда v8 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 BEEF==BEEF, что истинно.