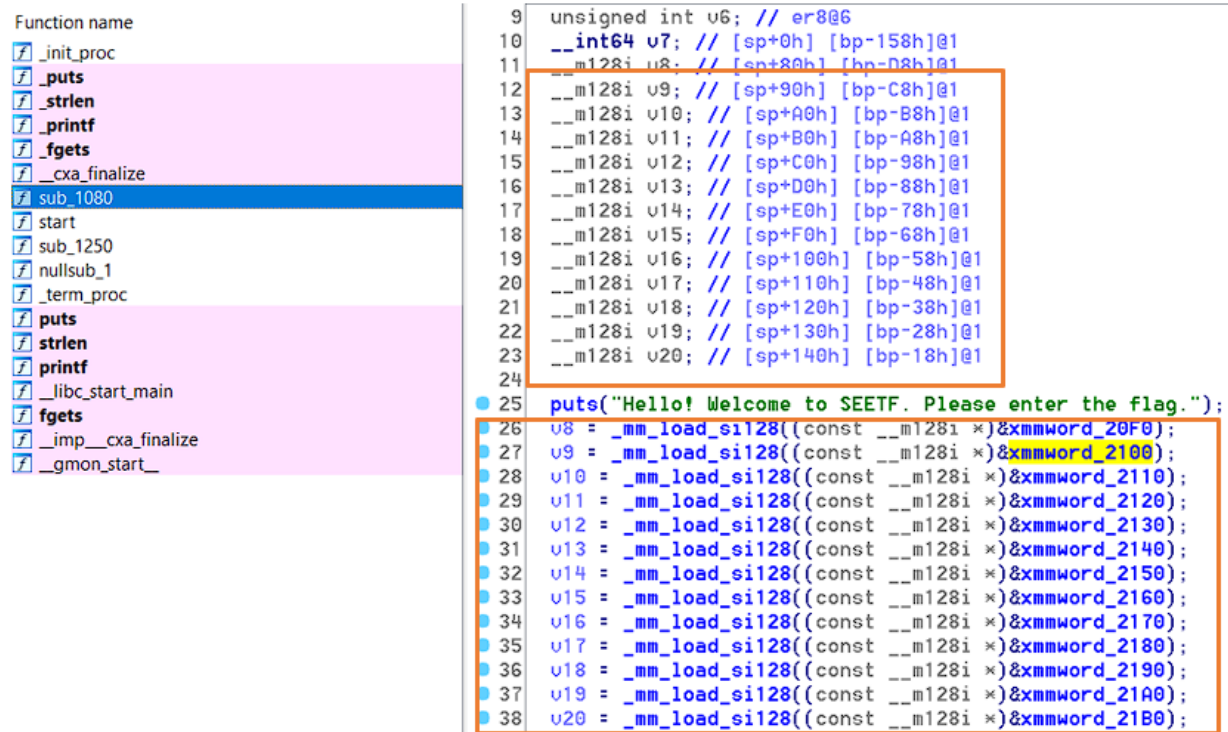


ASSIGNMENT 4

SETF_chall:

Sử dụng IDA Pro để xem mã giả.

Phần bắt đầu chương trình nằm ở sub_1080:



```
Function name
[?] _init_proc
[?] _puts
[?] _strlen
[?] _printf
[?] _fgets
[?] __cxa_finalize
[?] sub_1080
[?] start
[?] sub_1250
[?] nullsub_1
[?] _term_proc
[?] _puts
[?] _strlen
[?] _printf
[?] _libc_start_main
[?] _fgets
[?] __imp__cxa_finalize
[?] _gmon_start_

9  unsigned int v6; // er8@6
10  __int64 v7; // [sp+0h] [bp-158h]@1
11  __m128i v8; // [sp+80h] [bp-D8h]@1
12  __m128i v9; // [sp+90h] [bp-C8h]@1
13  __m128i v10; // [sp+A0h] [bp-B8h]@1
14  __m128i v11; // [sp+B0h] [bp-A8h]@1
15  __m128i v12; // [sp+C0h] [bp-98h]@1
16  __m128i v13; // [sp+D0h] [bp-88h]@1
17  __m128i v14; // [sp+E0h] [bp-78h]@1
18  __m128i v15; // [sp+F0h] [bp-68h]@1
19  __m128i v16; // [sp+100h] [bp-58h]@1
20  __m128i v17; // [sp+110h] [bp-48h]@1
21  __m128i v18; // [sp+120h] [bp-38h]@1
22  __m128i v19; // [sp+130h] [bp-28h]@1
23  __m128i v20; // [sp+140h] [bp-18h]@1
24
25  puts("Hello! Welcome to SEETF. Please enter the flag.");
26  v8 = _mm_load_si128((const __m128i *)&__xmmword_20F0);
27  v9 = _mm_load_si128((const __m128i *)&__xmmword_2100);
28  v10 = _mm_load_si128((const __m128i *)&__xmmword_2110);
29  v11 = _mm_load_si128((const __m128i *)&__xmmword_2120);
30  v12 = _mm_load_si128((const __m128i *)&__xmmword_2130);
31  v13 = _mm_load_si128((const __m128i *)&__xmmword_2140);
32  v14 = _mm_load_si128((const __m128i *)&__xmmword_2150);
33  v15 = _mm_load_si128((const __m128i *)&__xmmword_2160);
34  v16 = _mm_load_si128((const __m128i *)&__xmmword_2170);
35  v17 = _mm_load_si128((const __m128i *)&__xmmword_2180);
36  v18 = _mm_load_si128((const __m128i *)&__xmmword_2190);
37  v19 = _mm_load_si128((const __m128i *)&__xmmword_21A0);
38  v20 = _mm_load_si128((const __m128i *)&__xmmword_21B0);
```

Nhận thấy v8 tới v20 là 1 mảng. Chúng mang những giá trị đặc biệt, có thể sẽ có liên quan tới chuỗi bí mật cần tìm (tạm gọi là flag).

Phân tích đoạn mã giả tiếp theo:

```

139 fgets((char *)&v7, 128, stdin);
140 if ( strlen((const char *)&v7) == 53 )
141 {
142     puts("Good work! Your flag is the correct size.");
143     puts("On to the flag check itself...");
144     v1 = strlen((const char *)&v7);
145     v2 = 0LL;
146     v3 = v1 - 1;
147     do
148     {
149         v6 = v2;
150         if ( v3 == v2 )
151         {
152             puts("Success! Go get your points, champ.");
153             return 0LL;
154         }
155         v4 = v8.m128i_i32[v2];
156         v5 = v2 ^ ((*(_BYTE *)&v7 + v2) + 69);
157         ++v2;
158     }
159     while ( (_BYTE)v4 == (_BYTE)v5 );
160     printf("Flag check failed at index: %d", v6, v2);
161     result = 1LL;
162 }
163 else
164 {
165     printf("Flag wrong. Try again.", 128LL);
166     result = 1LL;
167 }
168 return result;
169 }

```

v7 là chuỗi nhập vào. Nếu chuỗi này có chiều dài là 52 (không tính kí tự NULL) thì sẽ so sánh từng kí tự trong v7 với flag bằng vòng lặp do-while.

Nếu đã xét xong kí tự cuối cùng thì thông báo đúng flag:

```

146     v3 = v1 - 1;
147     do
148     {
149         v6 = v2;
150         if ( v3 == v2 )
151         {
152             puts("Success! Go get your points, champ.");
153             return 0LL;
154         }
155         v4 = v8.m128i_i32[v2];
156         v5 = v2 ^ ((*(_BYTE *)&v7 + v2) + 69);
157         ++v2;

```

Trong từng vòng lặp, v2 là biến i, lấy ra từng kí tự của flag (v8) và thực hiện phép tính với v7 (input):

$$\text{flag}[i] = I \wedge (\text{input}[i] + 69) \quad (I)$$

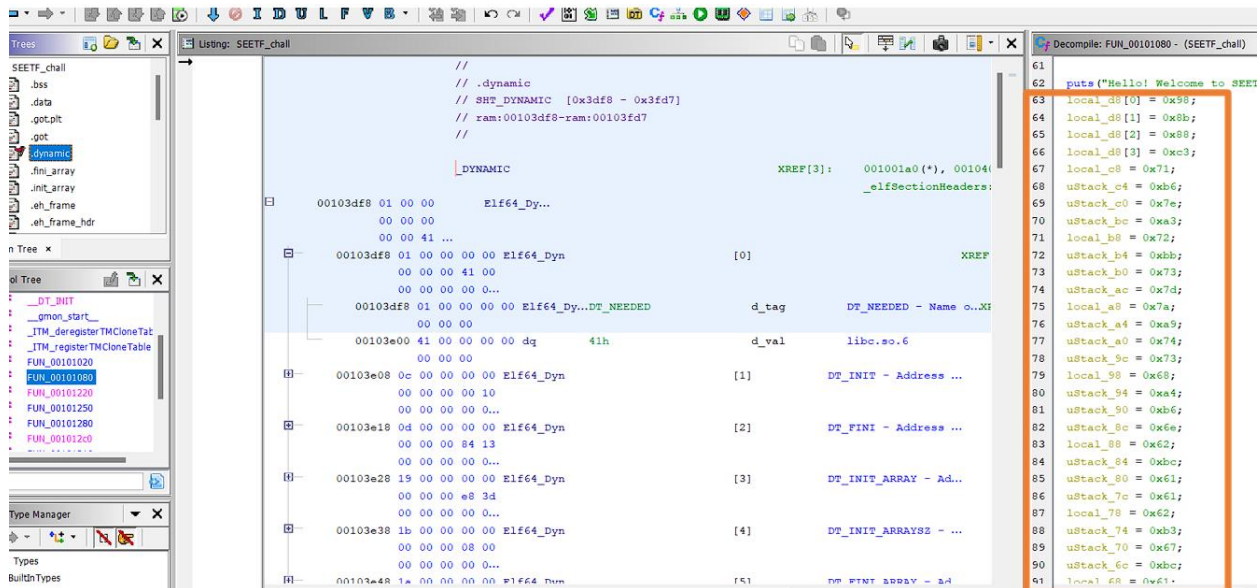
Sử dụng Ghidra để xem các giá trị của v8. Ta có:

v8[0] = 98h

v8[1] = 8bh

...

v8[51] = f1h



Từ phương trình (I), suy ngược lại:

$$\text{input}[i] = (\text{v8}[i] \wedge i) - 69$$

Từ đó input cần nhập là: SEE{0n3_5m411_573p_81d215e8b81ae10f1c08168207fba396}

```
(kali@kali)-[~/Downloads]
$ ./SEETF_chall
Hello! Welcome to SEETF. Please enter the flag.
SEE{0n3_5m411_573p_81d215e8b81ae10f1c08168207fba396}
Good work! Your flag is the correct size.
On to the flag check itself...
Success! Go get your points, champ.
```

Hard_chall:

Chạy thử chương trình:

```
(kali@kali)-[~/Downloads]
$ ./hard_chal
[$] Enter your input in the form: words_with_underscores_and_letters: dddddddd
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: dd_aa_bb
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: 
```

Ta muốn tượng được logic của chương trình:

Bước 1: Nhập chuỗi từ bàn phím

Bước 2: Kiểm tra chuỗi có giống format yêu cầu (các ký tự cách nhau bằng dấu gạch dưới). Nếu khác, quay lại bước 1. Nếu giống format, kiểm tra chuỗi nhập có giống với 1 chuỗi cho trước:

Nếu khác, thông báo “không giống” và quay lại bước 1.

Nếu giống, ta quan sát mã giả để xem hành động của chương trình.

Sử dụng IDA Pro để xem mã giả ở hàm main:

```
54 while ( v20 )
55 {
56     sub_10F0("[\$] Enter your input in the form: words_with_underscores_and_letters: ");
57     sub_1110("%s", &v36);
58     v21 = 0;
59     for ( i = 0; ; i = s((unsigned int)i) )
60     {
61         v9 = sub_10D0(&v36);
62         if ( !(unsigned __int8)l((unsigned int)i, (unsigned int)v9) )
63             break;
64         if ( (unsigned __int8)eq(*( (_BYTE *)&v43 + i - 1056), 95LL) ^ 1 )
65             v21 = s((unsigned int)v21);
66     }
67     v10 = sub_10D0(&v36);
68     if ( (unsigned __int8)ev((unsigned int)v10) ^ 1
69         || (v11 = sub_10D0(&v36),
70             v12 = su((unsigned int)v11, 1LL),
71             (unsigned __int8)eq(*( (_BYTE *)&v43 + v12 - 1056), 95LL))
72         || (unsigned __int8)v(&v36) ^ 1
73         || (unsigned __int8)ev((unsigned int)v21) ^ 1 )
74     {
75         sub_10C0("[\$] This won't do...");
76     }
77     else
78     {
79         LODWORD(v13) = sub_10D0(&v36);
80         v25 = v13 - 1;
81         v14 = 16 * ((v13 + 15) / 0x10uLL);
82         while ( &v19 != &v19 )
83             ;
84         v15 = alloca(v14 & 0xFFF);
85         if ( v14 & 0xFFF )
86             *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8) = *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8);
87         v26 = &v19;
88         sub_10B0(&v19, &v36, &v36, &v19);
```

Chương trình có 1 vòng while để kiểm tra chuỗi nhập vào (biến v36) rồi lặp lại nếu không đúng. Ở lệnh if dòng 68:

```

66 }
67 v10 = sub_10D0(&v36);
68 if ( (unsigned __int8)ev((unsigned int)v10) ^ 1
69 || (v11 = sub_10D0(&v36),
70     v12 = su((unsigned int)v11, 1LL),
71     (unsigned __int8)eq(((_BYTE *)&v43 + v12 - 1056), 95LL))
72 || (unsigned __int8)v(&v36) ^ 1
73 || (unsigned __int8)ev((unsigned int)v21) ^ 1 )
74 {
75     sub_10C0("[&v36] This won't do...");
76 }
77 else
78 {
79     LODWORD(v13) = sub_10D0(&v36);
80     v25 = v13 - 1;
81     v14 = 16 * ((v13 + 15) / 0x10uLL);
82     while ( &v19 != &v19 )
83     ;
84     v15 = alloca(v14 & 0xFFF);
85     if ( v14 & 0xFFF )
86         *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8) = *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8);
87     v26 = &v19;
88     sub_10B0(&v19, &v36, &v36, &v19);

```

Chương trình sẽ in ra “This won’t do” nếu chương trình đã so chuỗi nhập vào bằng hình thức nào đó. Nhớ lúc chạy thử chương trình, có thể hiểu đây là trường hợp nhập không đúng format.

Kiểm tra phần else:

Dòng 79 lưu kết quả trả về của hàm sub_10D0 có tham số là chuỗi nhập vào (v36) vào v13. Dòng 80 là v25 = v13 - 1. Điều này làm ta nhớ lại về hàm lấy chiều dài chuỗi, bỏ đi kí tự null cuối chuỗi. Có thể hiểu sub_10D0 là hàm strlen().

Dòng 81 có những con số liên quan tới thập lục phân (16, 15). Từ 84 tới 86 biến v14 được truyền cho hàm alloca, có thể đây là chiều dài của bộ nhớ cấp phát động cho v19.

```

81 v17 = 16 * ((v13 + 15) / 0x10uLL);
82 while ( &v19 != &v19 )
83 ;
84 v15 = alloca(v14 & 0xFFF);
85 if ( v14 & 0xFFF )
86     *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8) = *(__int64 *)((char *)&v19 + (v14 & 0xFFF) - 8);
87 v26 = (__int64)&v19;
88 sub_10B0(&v19, &v36, &v36, &v19);
89 for ( j = 0; ; j = encode((__int64)&v43, v26, j) )
90 {
91     v16 = sub_10D0(v26);
92     if ( !(unsigned __int8)l(j, v16) )
93         break;
94 }
95 v17 = sub_1100(v26, "odt_sjtfnb_jc_c_fiajb_he_ciu_hnkn_atvfjp");
96 if ( (unsigned __int8)eq((unsigned int)v17, 0LL) )
97 {
98     sub_10C0("[&v36] Correct!");
99     v20 = 0;
100 }
101 else
102 {
103     sub_10C0("[&v36] Incorrect...");
104 }
105 }
106 }
107 v27 = 117;
108 v28 = 105;
109 v29 = 117;
110 v30 = 99;
111 v31 = 117;

```

Cho tới khi lần cuối v19 xuất hiện, ta thấy nó chỉ có 1 tác động đến chương trình. Đó là ở dòng 87: v26 bằng địa chỉ v19, mà dòng v88 là 1 hàm được truyền 2 tham số là v19 và chuỗi nhập vào. Từ đó có thể hiểu v26 có liên quan tới chuỗi nhập vào qua hàm sub_10B0

```

;
) sub_10B0      proc near                                ; CODE XREF: main+23D↓p
)              rep nop edx                               |
;              repne jmp cs:strcpy_ptr

```

Nhận thấy sub_10B0 là hàm strcpy, liên kết lại và đúc kết rằng v26 = chuỗi nhập vào.

Dòng 89 – 100:

```

88      sub_10B0(v19, v26, v26, v19),
89      for ( j = 0; ; j = encode((__int64)&v43, v26, j) )
90      {
91          v16 = sub_10D0(v26);
92          if ( !(unsigned __int8)l(j, v16) )
93              break;
94      }
95      v17 = sub_1100(v26, "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp")
96      if ( (unsigned __int8)eq((unsigned int)v17, 0LL) )
97      {
98          sub_10C0("[&] Correct!");
99          v20 = 0;
100     }

```

sub_1100 là hàm strcmp

```

0
0 sub_1100      proc near                                ; CODE XREF: main+2A0↓p
0              rep nop edx
4              repne jmp cs:strcmp_ptr
..
;              ; DATA XREF: .got:__libc_start_main_ptr↑o
int strcmp(const char *s1, const char *s2)
    extrn strcmp:near ; DATA XREF: .got:strcmp_ptr↑o
    extrn __isoc99_scanf:near ; DATA XREF: .got:__isoc99_scanf_ptr↑o
    extrn __cxa_finalize:near ; weak
    extrn __cxa_finalize:near ; DATA XREF: .got:__cxa_finalize_ptr↑o
    extrn _ITM_deregisterTMCloneTable ; weak
    extrn _ITM_deregisterTMCloneTable:near ; DATA XREF: .got:_ITM_deregisterTMCloneTable_ptr↑o
    extrn __gmon_start__:near ; weak
    extrn __gmon_start__:near ; CODE XREF: _init_proc+14↑p
    extrn __gmon_start__:near ; DATA XREF: .got:__gmon_start___ptr↑o
    extrn _ITM_registerTMCloneTable ; weak
    extrn _ITM_registerTMCloneTable:near ; DATA XREF: .got:_ITM_registerTMCloneTable_ptr↑o

    end _start

```

Dòng 95 (hàm main), strcmp sẽ so sánh 2 chuỗi đầu vào, trả về 0 nếu giống nhau, còn lại trả về 1. Kết quả trả về lưu vào v17.

```

95     v17 = sub_1100(v26, "odt_sjtfnb_jc_c_fiajb_he_ciuh_nkn_atvfjp")
96     if ( (unsigned __int8)eq((unsigned int)v17, 0LL) )
97     {
98         sub_10C0("[&] Correct!");
99         v20 = 0;

```

Dòng 96, so sánh v17 với 0LL bằng hàm eq. Xem qua hàm eq:

```

1 int __fastcall eq(int a1, int a2)
2 {
3     int result; // eax@3
4
5     __asm { rep nop edx }
6     if ( a1 || a2 )
7     {
8         if ( a1 && a2 )
9             result = eq((unsigned int)(a1 - 1), (unsigned int)(a2 - 1));
10        else
11            result = 0;
12    }
13    else
14    {
15        result = 1;
16    }
17    return result;
18}

```

Hàm eq này khá thú vị, sẽ giảm giá trị a và b dần dần để biết chúng bằng nhau hay không. Nếu a = b: trả về 1, ngược lại trả về 0.

Vậy có thể hiểu rằng v26 so sánh với chuỗi “odt_..._atvfjp”. Nếu bằng sẽ in ra correct. Nhưng v26 đã được biến đổi trong hàm for trước đó. Ở đây ta có 3 hàm cần làm rõ là encode(), su() và hàm l ().

Đầu tiên là hàm su:

```

1 int __fastcall su(int a1, unsigned int a2)
2 {
3     int result; // eax@2
4
5     __asm { rep nop edx }
6     if ( (unsigned __int8)eq(a2, 0LL) )
7     {
8         result = a1;
9     }
10    else if ( (unsigned __int8)eq((unsigned int)a1, 0LL) )
11    {
12        result = 0;
13    }
14    else
15    {
16        result = su((unsigned int)(a1 - 1), a2 - 1);
17    }
18    return result;
19 }

```

Logic hàm su:

Nếu số thứ nhất $a1 > \text{số thứ 2 } a2$: trả về $a1 - a2$

Nếu không: trả về 0.

Chạy thử chương trình mô phỏng hàm su bằng C (đổi eq bằng dấu ==):

```

9  #include <stdio.h>
10 int su(int a1, unsigned int a2)
11 {
12     int result; // eax@2
13     if ( a2 == 0 )
14     {
15         result = a1;
16     }
17     else if ( a1 == 0 )
18     {
19         result = 0;
20     }
21     else
22     {
23         result = su((unsigned int)(a1 - 1), a2 - 1);
24     }
25     return result;
26 }
27 int main()
28 {
29     printf(" su1: %d\n su2: %d\n su3: %d\n su4: %d\n su5: %d\n", su(3, 4), su(7, 3), su(1, 0), su(0, 1), su(3, 3));
30
31     return 0;
32 }
33

```

input

```

su1: 0
su2: 4
su3: 1
su4: 0
su5: 0

```

Trong hàm encode, hàm su xuất hiện dạng su(biến a, 1), tương đương $a=1$ trong C.

Tiếp theo là hàm l:


```

1 int __fastcall l(unsigned int a1, unsigned int a2)
2 {
3     int result; // eax@3
4     int v3; // ebx@8
5     int v4; // eax@8
6
7     __asm { rep nop edx }
8     if ( (unsigned __int8)eq(a1, 0LL) && (unsigned __int8)eq(a2, 0LL) )
9     {
10         result = 0;
11     }
12     else if ( (unsigned __int8)eq(a2, 0LL) )
13     {
14         result = 0;
15     }
16     else if ( (unsigned __int8)eq(a1, 0LL) )
17     {
18         result = 1;
19     }
20     else
21     {
22         v3 = su(a2, 1LL);
23         v4 = su(a1, 1LL);
24         result = 1((unsigned int)v4, (unsigned int)v3);
25     }
26     return result;
27 }

```

Logic hàm l (a1, a2):

Nếu a1 = 0 và a2 = 0: trả về 0

Ngược lại, nếu chỉ a2 = 0: trả về 0

Ngược lại, nếu chỉ a1 = 0: trả về 1

Ngược lại: gọi đệ quy hàm l (a1-1, a2-1).

Hàm này na ná hàm su() khi so sánh a1 và a2 bằng việc giảm dần giá trị cả 2 biến, hàm l() sẽ trả về 1 nếu a1 < a2, ngược lại là 0. (có thể hiểu hàm l là hàm less)

Kế đến là hàm s:

```

1 __int64 __fastcall s(int a1)
2 {
3     __int64 v2; // [sp-8h] [bp-8h]@1
4
5     __asm { rep nop edx }
6     *((_DWORD *)&v2 - 1) = a1;
7     return (unsigned int)(*((_DWORD *)&v2 - 1) + 1);
8 }

```

Logic hàm s(a): tương đương a++ trong C.

Rồi hàm a(a1, a2):

```
1 int __fastcall a(int a1, unsigned int a2)
2 {
3     int result; // eax@2
4     unsigned int v3; // ebx@3
5     int v4; // eax@3
6
7     __asm { rep nop edx }
8     if ( (unsigned __int8)eq(a2, 0LL) )
9     {
10         result = a1;
11     }
12     else
13     {
14         v3 = su(a2, 1LL);
15         v4 = s(a1);
16         result = a(v4, v3);
17     }
18     return result;
19 }
```

Logic hàm a:

Nếu a2 = 0: trả về a1

Ngược lại, gọi đệ quy a(a1 + 1, a2 - 1) (như đã phân tích hàm su và hàm s ở trên)

Khi a2 giảm 1 lượng x, đồng nghĩa a1 tăng 1 lượng x. Khi a2 = 0 (tức x = a2) thì trả về a1 + a2. Vậy hàm a(a1, a2) sẽ trả về tổng a1 và a2. (hiểu hàm a là hàm add).

Cuối cùng là hàm m:

```
1 __int64 __fastcall m(int a1, unsigned int a2)
2 {
3     __int64 result; // rax@2
4     unsigned int v3; // eax@3
5
6     __asm { rep nop edx }
7     if ( (unsigned __int8)eq(a2, 0LL) )
8     {
9         result = 0LL;
10    }
11    else
12    {
13        v3 = m((unsigned int)a1, a2 - 1);
14        result = (unsigned int)a(a1, v3);
15    }
16    return result;
17 }
```

Ta đã có hàm su (trừ - subtract), hàm a (cộng - add), và giờ là hàm m là multiply (phép nhân)

Logic hàm m:

Nếu $a2 = 0$: trả về 0.

Ngược lại, trả về tổng của $a1$ và $m(a1, a2 - 1)$.

Cho tới khi $a2 = 0$ để $m(a1, 0)$ trả về 0, hàm m sẽ có $a2$ lần cộng $a1$, tương đương tích của $a1$ và $a2$.

Như thế ta đã phân tích đủ các hàm xuất hiện trong hàm encode. Phân tích hàm encode:

Tại hàm main, encode được truyền 3 tham số là $v43$, $v26$ và j

```
89 |         for ( j = 0; ; j = encode((__int64)&v43, v26, j) )
90 |         {
91 |             v16 = sub_10D0(v26);
```

Xuyên suốt hàm encode, tham số đầu tiên không được sử dụng, tham số $v26$ là chuỗi nhập vào, j là index.

Dòng 25 – 27:

```
24 | v22 = a1;
25 | for ( i = a3; (unsigned __int8)eq((__BYTE *) (i + a2), 95LL); i = s((unsigned int)i) )
26 | ;
27 | for ( j = s((unsigned int)i); (unsigned __int8)eq((__BYTE *) (j + a2), 95LL); j = s((unsigned int)j) )
28 | ;
```

2 vòng for trên thực chất sẽ so sánh từng ký tự với dấu gạch dưới (số 95 trong ASCII), bắt đầu từ bên trái, nếu giống thì xét ký tự tiếp theo bên phải cho tới khi gặp ký tự khác ký tự gạch dưới. Ký tự khác ký tự gạch dưới này (tạm gọi $k1$) sẽ lưu vào $v11$, rồi ký tự thứ 2 bên phải ký tự $k1$ mà khác với ký tự gạch dưới sẽ lưu vào $v12$. Ví dụ: $a2 = "a_d"$ thì $v11$ lưu vị trí của a , $v12$ lưu vị trí của d .

```
29 | v11 = *(__BYTE *) (i + a2);
30 | for ( k = 0; (unsigned __int8)::1((unsigned int)k, 5LL); k = s((unsigned int)k) )
31 | {
32 |     for ( l = 0; (unsigned __int8)::1((unsigned int)l, 5LL); l = s((unsigned int)l) )
33 |     {
34 |         v3 = ::m((unsigned int)k, 5LL);
35 |         v4 = a((unsigned int)v3, (unsigned int)l);
36 |         if ( (unsigned __int8)eq((unsigned int)aAbcdefghijklmn[v4], (unsigned int)v11) )
37 |         {
38 |             v14 = k;
39 |             v15 = l;
40 |         }
41 |     }
42 | }
43 | v12 = *(__BYTE *) (j + a2);
```

Thoạt đầu nhìn tưởng 2 vòng for từ dòng 30 đến 32 là ma trận 5×5 , nhưng nhìn kỹ lại thì nó tính

$$\text{index} = k * 5 + l$$

và như thế i thuộc $[0;25]$ có thể biểu diễn 25 phần tử của biến $aAbcdefghijklmn$, vốn là 25 chữ cái alphabet (không tính chữ q).

```
db      0
aAbcdefghijklmn db 'abcdefghijklmnopqrstuvwxyz',0
; DATA XREF: encode+D9↑o
; encode+199↑o
```

Nếu $aAbcdefghijklmn[\text{index}] =$ ký tự đang xét ($v11$) thì lưu lại vị trí trong bảng chữ cái thông qua k ($v14$) và l ($v15$) ($\text{index} = v14 * 5 + v15$).

Tương tự với kí tự xét thứ 2 (v12), lưu vị trí thông qua v18 và v19.

```
for ( m = 0; (unsigned __int8)::1((unsigned int)m, 5LL); m = s((unsigned int)m) )
{
    for ( n = 0; (unsigned __int8)::1((unsigned int)n, 5LL); n = s((unsigned int)n) )
    {
        v5 = ::m((unsigned int)m, 5LL);
        v6 = a((unsigned int)v5, (unsigned int)n);
        if ( (unsigned __int8)eq((unsigned int)aAbcdefghijklmn[v6], (unsigned int)v12) )
        {
            v18 = m;
            v19 = n;
        }
    }
}
```

Phần cuối cùng của hàm encode là thay đổi giá trị chuỗi đầu vào (a2, hay v26):

```
v7 = ::m((unsigned int)v14, 5LL);
*( _BYTE *) (a2 + i) = aVastbcdefghijk[a((unsigned int)v7, (unsigned int)v19)];
v8 = ::m((unsigned int)v18, 5LL);
*( _BYTE *) (a2 + j) = aCornfieldsabgh[a((unsigned int)v8, (unsigned int)v15)];
return s((unsigned int)j);
```

Kí tự đầu tiên không phải kí tự gạch dưới thay đổi thành aVastbcdefghijk[v14 * 5 + v19]

Kí tự thứ hai sau kí tự đầu tiên không phải kí tự gạch dưới thay đổi thành aCornfieldsabgh[v18 * 5 + v15]

Ở hàm main, vòng for dòng 89 sẽ chạy tới khi duyệt xong kí tự cuối cùng của v26:

```
89     for ( j = 0; ; j = encode((__int64)&v43, v26, j) )
90     {
91         v16 = sub_10D0(v26);
92         if ( !(unsigned __int8)l((unsigned int)j, (unsigned int)v16) )
93             break;
94     }
95     v17 = sub_1100(v26, "odt_sjtfnb_jc_c_fiajb_he_ciuh_nkn_atvfjp");
```

rồi so sánh chuỗi của v26 sau khi encode có giống với chuỗi “odt_sjtfnb_jc_c_fiajb_he_ciuh_nkn_atvfjp”

o trong aVast là $17 = 3 * 5 + 2 \Rightarrow v14 = 3, v19 = 2$

d trong aCorn là $8 = 1 * 5 + 3 \Rightarrow v18 = 1, v15 = 3$

$\Rightarrow v14 * 5 + v15 = 3 * 5 + 3 = 18$, trong alphabet là t

Cứ tiếp tục dịch ngược, kết quả là đoạn v26 cần tìm là the_inside_of_a_field_of_corn_and_dreams

Chạy thử:

```
[!] Enter your input in the form: words_with_underscores_and_letters: the_inside_of_a_field_of_corn_and_dreams
[!] Correct!
[!] uiuctf{the_inside_of_a_field_of_corn_and_dreams}
```