


Angstrom CTF 2021

Competition Intro

This competition was running from 2nd April to 7th April and its rating was 46.09 while I was playing this CTF.

Totally, there are 11 challenges in RE category with the level from newbies to advance. I solved 6 challenges during the CTF took place but I feel a little uneasy because I didn't solve the ***lambda lambda lambda*** challenge.

Jailbreak

 *Challenge's description:* Clam was arguing with kmh about whether including 20 pyjails in a ctf is really a good idea, and kmh got fed up and locked clam in a jail with a python! [Can you help clam escape?](#)
Source file được tìm thấy tại [đây](#) nếu server down

Phân tích src và workflow của challenge

Hiểu tổng quát về chương trình

```
1  angstromCTF_2021:/>$../jailbreak
2
3  Welcome to clam's daring jailbreak! Please keep your hands and feet inside
4  What would you like to do?
```

- Thử running chương trình, ta sẽ thấy một dòng thông báo thể hiện ý tưởng của tác giả cho challenge: chúng ta phải làm gì đó để giải cứu cho clam đang bị kmh nhốt vào tù :<
- Thực hiện phân tích file ELF bằng IDA, dưới đây là một đoạn code được cắt từ cửa sổ decompile của IDA Pro ver:

```
1  int main()
2  {
3
4  //...[REDACTED]..
5
6  sub_1620(1LL);
7  while ( 1 )
8  {
9      v10 = sub_15A0(0);
10     puts(v10);
```

```

11     free(v10);
12     if ( !fgets(v31, 256, stdin) )
13         goto LABEL_24;
14     v31[strcspn(v31, "\n")] = 0;
15     if ( v5 )
16         break;
17     v6 = sub_15A0(2);
18     v7 = strcmp(v31, v6);
19     free(v6);
20     if ( v7 )
21     {
22         v17 = sub_15A0(6);
23         v18 = strcmp(v31, v17);
24         free(v17);
25         if ( !v18 )
26         {
27             sub_1620(7LL);
28 LABEL_24:
29             sub_1620(5LL);
30             goto LABEL_25;
31         }
32
33     //...[REDACTED]..
34
35 }

```

- Chúng ta sẽ thấy một vài điểm khá thú vị:
 - 2 hàm **sub_15A0()** và **sub_1620()** được gọi lặp đi lặp lại nhiều lần trong main
 - ngoài ra **sub_1620()** thật chất là sự kết hợp của **put()** và **sub_15A0()**, trong đó **sub_15A0()** là hàm nhận vào một số nguyên và trả về một chuỗi:

Phân tích hàm sub_15A0()


```

v1 = dword_4080[a1];
v2 = dword_4080[a1 + 1] - v1;
v3 = malloc(v2 + 1);
v3[v2] = 0;
v4 = v3;
if ( v2 > 0 )
{
    v5 = a1;
    v6 = 0LL;
    do
    {
        v7 = v5 ^ byte_4100[v1 + v6];
        v4[v6++] = v7;
        v5 = v5 * v7 + 17 * a1;
    }
    while ( v2 != v6 );
}
return v4;

```

pseudo code ban đầu của hàm sub_15A0

- Sau khi phân tích hàm này, ta sẽ thấy:
 - có 2 mảng **dword_4080** và **byte_4100**, dễ thấy **dword_4080** sẽ chứa index đầu tiên (**v1**) của tham số **a1** trong mảng lớn **byte_4100** và **v2** là index cuối
 - Sử dụng IDA Python để trích xuất dữ liệu 2 mảng trên từ file thực thi và sau đó re-implement lại hàm **sub15A0(int a1)**, ta sẽ sinh ra tất cả các chuỗi được trả về với tham số a1 tương ứng

 Chúng ta có thể xác định được các giá trị của tham số a1, vì $0 \leq a1 < \text{size}(\text{mảng } \text{dword_4080}) - 1$ (vì có $v2 = \text{dword_4080}[a1 + 1] - v1$)

- IDAPython Script ở [đây](#)
- Output từ IDA:

```

1  0 What would you like to do?
2  1 Welcome to clam's daring jailbreak! Please keep your hands and feet insi
3  2 look around
4  3 You look around your cell and you see an old bed along with a snake. Out
5  4 You're speaking nonsense. Cut that out.
6  5 It seems that clam won't be escaping today.
7  6 sleep
8  7 You lie down on the bed and close your eyes. You then get bitten by the
9  8 You look around your cell and you see an old bed along with a snake. Out
10 9 knock on the wall
11 10 You here a muffled voice from the other side saying "I shouldn't have b
12 11 pry the bars open
13 12 You start prying the prison bars open. Realizing this is unintended, km
14 13 You start prying the prison bars open. A wide gap opens and you slip th
15 14 throw the snake at kmh
16 15 pick the snake up
17 16 You pick the snake up.
18 17 You throw the snake at kmh and watch as he runs in fear.
19 18 You look around and see that kmh has already made the jail contrived! T
20 19 press the red button
21 20 press the green button
22 21 You pressed the red button. Nothing changed.
23 22 You pressed the green button. Nothing changed.
24 23 bananarama
25 24 For some reason, a flag popped out of the wall, and you walk closer to
26 25 flag.txt
27 26 r
28 27 Couldn't find flag file.
29 28 Attached to the flag is a key to the front door. It looks like clam is
30 29
31 30


```

Passing this challenge


- Sau khi đã có các chuỗi trả về từ hàm **sub15A0** với các tham số tương ứng, ta sẽ dễ dàng hiểu chương trình hơn.
- Tóm tắt workflow:
 - Thực hiện hành động thích hợp để giải cứu **clam**
 - Ngoài hành động cuối cùng "**bananarama**" là, còn phải set một số bằng 1337(bằng cách thực hiện hành động "press the red button", "press the green button" số

lần thích hợp) thì mới mở được flag.

```
1  Thứ tự các hành động như sau:  
2  
3  pick the snake up  
4  throw the snake at kmh  
5  pry the bars open  
6  press the red button  
7  press the green button  
8  press the red button  
9  press the red button  
10 press the green button  
11 press the green button  
12 press the green button  
13 press the red button  
14 press the red button  
15 press the green button  
16 bananarama
```

 Vì flag challenge này trên server, vì vậy để test có thể tạo file flag.txt ở local để test.

Infinity Gauntlet

-  Challenge's Description: All **clam** needs to do is snap and finite will turn into **infinite**...

```
InfinityGauntlet:/>$./infinity_gauntlet
Welcome to the infinity gauntlet!
If you complete the gauntlet, you'll get the flag!
=== ROUND 1 ===
foo(64832, 352) = ?
_
```

TL; DR

- Trả lời đúng các giá trị còn thiếu trong các phép toán mà chương trình đưa ra(bởi hai hàm **foo()** và **bar()**)

```
__int64 __fastcall bar(int a1, int a2, int a3)
{
    return (unsigned int)(a1 + a2 * (a3 + 1));
}
```

cách tính của hàm foo

```
1 __int64 __fastcall foo(int a1, int a2)
2 {
3     return a1 ^ (a2 + 1) ^ 0x539u;
4 }
```

cách tính của hàm bar

- Nội dung flag sẽ được đọc vào, và nó đã được tính toán qua một bước khởi tạo (các ký tự flag được xor lần lượt với [0, 17, 34, 51...], ta gọi mảng này là *i_xor*) trước khi đi vào vòng lặp chứa foo, bar (tên mặc định do IDA tạo đã được thay đổi để dễ dàng cho phân tích)

```
fgets(flagContent, 256, v7);
fclose(v8);
len_flag = strlen(flagContent, "\n");
_len_flag = len_flag;
flagContent[len_flag] = 0;
if ( len_flag )
{
    v10 = flagContent;
    v11 = 17 * len_flag;
    v12 = 0;
    do
    {
        *v10 ^= v12;
        v12 += 17;
        ++v10;
    }
    while ( (_BYTE)v12 != v11 );
}
```

các ký tự flag được xor lần lượt với i_xor = [0, 17, 34, 51...]

- Như vậy, sau khi tìm được flag, ta phải thực hiện thêm bước xor khởi tạo này để lấy lại flag ban đầu

```

77 LABEL_11:
78 printf("=== ROUND %d ===\n", (unsigned int)round_Rth);
79 result = rand();
80 if ( round_Rth <= 49 )
81 {
82     v18 = rand();
83     v19 = (unsigned __int16)(v18 + ((unsigned __int64)v18 >> 48)) - ((unsigned int)(v18 >> 31) >> 16);
84     goto LABEL_13;
85 }
86 do
87 {
88     v19 = flagContent[_len_flag & result] | ((unsigned __int8)((_len_flag & result) + round_Rth) << 8);

```

- Chú ý, sau 49 round đầu(49 câu hỏi phép tính đầu), byte cuối biến v19(gồm 2 byte, đóng vai trò như là kết quả các phép toán) sẽ chứa một byte **flagContent[unknow]**(với *unknow* là một số random, vì result= rand() ở dòng 79)
- Từ **flagContent[unknow]**, ta có thể khôi phục lại **flag[unknow]** bằng: **flag[unknow] = i_xor[unknow]^flagContent[unknow]**

Solve Script



- Trong bài này, ta sẽ sử dụng thêm thư viện pwn của bộ công cụ **pwntools** để tự động tương tác với server
- Vì ban đầu không biết được độ dài flag nên phải tìm đến khi nào mà ký tự '}' xuất hiện, lúc đó mới xác định được length của flag, trong script này, vì đã biết độ dài flag nên mình set dừng ngay sau khi đã tìm đủ tất cả các phần tử flag luôn.

```

Desktop:/>$python3 solve_InfinityGauntlet.py
[+] Opening connection to shell.actf.co on port 21700: Done
actf{snapped_away_the_end}
Desktop:/>$_

```

Bạn có thể tìm thấy source của challenge ở [đây](#) nếu không thể truy cập đến file đặt ở server, ngoài ra đây là một solve [script](#) tham khảo


Revex

- Chúng ta nhận được một dãy các **regex** pattern từ tác giả như sau:

```
^(?=.*re)(?={21}[^_]{4}\}$)(?={14}b[^_]{2})(?={8}[C-L])(?={8}[B-F])(?={
```

- Để có thể solve tách từng pattern → mỗi pattern sẽ chứa một điều kiện. **FLAG** là chuỗi thỏa mãn tất cả các regex pattern này:

```
1  ^(?=.*re)
2  (?={21}[^_]{4}\}$)
3  (?={14}b[^_]{2})
4  (?={8}[C-L])
5  (?={8}[B-F])
6  (?={8}[^B-DF])
7  (?={7}G(?<pepega>...){7}t\k<pepega>)
8  (?=.*u[^z].$)
9  (?={11}(?<pepeega>[13])s.{2}(?! \k<pepeega>)[13]s)
10 (?=.*_{2}_)
11 (?=actf\{)
12 (?={21}[p-t])
13 (?=.*1.*3)
14 (?={20}(?=.*u)(?=.*y)(?=.*z)(?=.*q)(?=.*_))
15 (?=.*Ex)
```

 Để đơn giản, gọi tên các pattern bằng STT dòng như hiển thị ở trên, ngoài ra format mặc định của flag là **actf{some-strings}**


- Xét 3 pattern p(4), p(5) và p(6):
 - p(4) ký tự thứ 9 của chuỗi nằm trong đoạn C-L
 - p(5) ký tự thứ 9 của chuỗi nằm trong đoạn từ B-F
 - Suy ra: Ký tự nằm trong đoạn C-F (có 4 khả năng C, D, E, F), lại có p(6): ký tự thứ 9 khác {B, D, F}
 - Vậy ký tự thứ 9 là E**

- Theo như (7) thì ký tự thứ 8 là **G**, p(1) và p(15) nói rằng trong flag phải chứa "re" và "Ex"
 - Từ các dữ kiện trên, ta dự đoán từ đầu tiên sau chuỗi "**actf{**" là "**reGEx**"
- Cũng từ p(7):
 - 2 ký tự thuộc group được đặt tên *pepega* là "**Ex**", sau lần xuất hiện đầu tiên thì sau 7 ký tự tùy chọn và ký tự **t** thì Ex lại xuất hiện lần nữa, ta có:
 - actf{reGEx * * * * * tEx...
- Xét p(2): **21** ký tự đầu tiên + **4 ký tự cuối** (khác "_" + "}"), suy ra len(flag) là 26:

actf{reGEx * * * * * tEx*****}
 - pattern (14): (?.{20})(?.*u)(?.*y)(?.*z)(?.*q)(?.*_)) - sau 20 ký tự thì 5 ký tự cuối(không tính "}") sẽ thuộc { u, y, z, q, _} nhưng vì 4 ký tự khác "_", **vì vậy nên ký tự 21 phải là "_"**
 - p(12): (?.{21}[p-t]) - suy ra **ký tự 22 phải là q**
 - p(8): (?.*u[^z].\$): thì 3 ký tự cuối là "**u***", nhưng * khác z, suy ra * phải là y
 - suy ra: actf{reGEx * * * * * tEx_qzuy}
- Theo p(9), ký tự thứ 12 sẽ thuộc {1,3}sau đó là **s**, lại có p(13): (?.*1.*3) - ta sẽ có thứ tự xuất 1 đến 3, nên **ta suy đoán ký tự thứ 12 là "1", "1s"**, quá hợp lý luôn
 - Lại có p(10): (?.*_.{2}_) -> 2 ký tự nằm giữa 2 "_", có vẻ "**_1s_**" là hợp lí.
 - Và theo p(3): (?.{14}b[^_]{2}) ==> ký tự thứ 15 là "b". Lúc này ta có:
 - actf{reGEx_1s_b * * tEx_qzuy}
 - Theo p(9): thì ký tự 16 thuộc group **pepega={1, 3}**, nhưng khác lần xuất hiện đầu tiên => "**3**", sau đó là **s**

Flag: actf{reGEx_1s_b3stEx_qzuy}

Lockpicking

 Challenge's Description: Clam wanted to have a lockpicking competition but decided against it. So, he settled for a [digital lockpicking competition](#) instead.

Nếu không thể truy cập vào file source của server, bạn có thể tìm thấy nó ở [đây](#)

TL; DR

1. Cần input để có thể mở khóa: "**The lock opens**" và cùng với thông báo này, file **flag.txt** sẽ được mở để print flag. Đây cũng lại là flag đặt ở server, vì vậy để test(trong trường hợp server down) thì tạo một file flag.txt ở local
2. Các ký tự trong input chỉ thỏa mãn khi thuộc tập hợp{!, ?, @, I, M, O, i, m, o}. Lượng input được nhập là giới hạn và một mảng **a** có 32 phần tử sẽ thay đổi dựa vào mỗi ký tự trong input
3. Sau khi xử lý thì ptu từ **a[14]** đến **a[18]** phải khác 0, thỏa mãn, thì flag được in ra

Theo ý kiến cá nhân của mình, thì bài này thuần về suy luận logic hơn, cũng chính vì lí do đó mà với tư duy vừa gà vừa chậm của mình thì phải mất khá lâu (hơn một ngày) mình mới hiểu được cách hoạt động, cũng như tìm được một input thỏa mãn

Các bước giải

Hiểu work-flow của chương trình

- Dưới đây là 3 hàm chính của chương trình hàm main, hàm **eval_v8Array**(hàm sub15A0) và **input_basedFunc**(hàm sub1690):
 - hàm main()

- `eval_v8Array()` - đây là hàm tính toán giá trị các phần tử của mảng(ở đây đặt tên là `v8Array`) dựa vào các ký tự input đầu vào
- `input-basedFunc()` - ở trên ta có nói mảng tính toán dựa vào input đầu vào, thực ra là thông qua hàm này, với mỗi ký tự sẽ có một case thích hợp, đặt biệt các ký tự `i`, `m`, `o`, `l`, `M`, `O` và `!` cũng sẽ tác động trực tiếp đến phần tử của mảng **`v8Array`**(`v8Arr[0]`, `v8Arr[4]`, `v8Arr[8]`, `v8Arr[13]`)

```

1  int main(__int64 a1, char **input_i, char **a3)
2  {
3      char *ptrInput; // rbx
4      __int64 *v4; // rax
5      __int64 result; // rax
6      unsigned __int64 v6; // rt1
7      int len_input; // [rsp+Ch] [rbp-45Ch]
8      __int64 v8Arr; // [rsp+10h] [rbp-458h]
9      char v9; // [rsp+1Ch] [rbp-44Ch]
10     char input; // [rsp+30h] [rbp-438h]
11     unsigned __int64 v11; // [rsp+438h] [rbp-30h]
12
13     v11 = __readfsqword(0x28u);
14
15     sub_55FDF1FE1390((__int64)&v8Arr);           // khởi tạo cho v8
16     len_input = 0;
17     while (true)
18     {
19         if (v8[12])                             // v10 khởi tạo = 0, v10
20             suspiciousFun1((signed int *)&v8Arr); // CHÚ Ý HÀM NÀY
21
22         printf("> ", input_i);
23
24         if ( !fgets(&input, 1024, stdin) )        // get input thất bại
25         {
26             puts("You walk away from the lock.");
27             goto FAIL_LABEL;
28         }
29
30         int i = 0
31         while(ptrInput[i] == '\0' || ptrInput[i] == '\n')
32         {
33             if (input_basedFunc(&v8Arr, ptrInput[i], &len_input) ) {
34                 puts("Your actions prove to be ineffective against the lock.");
35                 goto FAIL_LABEL;
36             }
37             i++;
38         }
39         // v7 > 164 cũng thất bại
40         if ( len_input > 164 )

```

```

41     break;
42
43     v4 = &v8Arr; // v4 tham chiếu đến địa c
44     while (v4[14]) // check v8[14]
45     {
46         v4 = v4 + 1;
47         if (v8[5] == v4) // vì v9 nằm ở rsp + 0x15, v8(v4 được gán bằng v8) n
48             // thì trong khi v8[14] !=
49         {
50             puts("The lock opens.");
51             getFlag();
52             goto FAIL_LABEL;
53         }
54     }
55 }
56 puts("Your hands grow weary from the stiffness of the lock.");
57 FAIL_LABEL:
58 v6 = __readfsqword(0x28u);
59 result = v11 - v6;
60 if ( v11 == v6 )
61     result = 0LL;
62 return result;
63 }

```

```

1  int eval_v8Array(__int64 v8)
2  {
3      int ele_4; // ecx
4      int ele_8; // edx
5      char ele_15; // al
6      char ele_13; // si
7      signed __int64 result; // rax
8
9      ele_4 = v8[4];
10     ele_8 = v8[8];
11     if ( ele_4 == 11 && v8[0] == 22 )
12         v8[18] = ele_8 == 7; // *
13     ele_15 = v8[15];
14     ele_13 = v8[13];
15     if ( ele_8 == 7 )
16     {
17         if (!ele_15 || (ele_15 = v8[18] == 0 )
18         {
19             ele_15 = 0;
20             if ( ele_4 == 8 && v8[0] == 9 )
21             {
22                 ele_15 = v8[16];
23                 if ( ele_15 )
24                     ele_15 = ele_13 ^ 1;

```

```

25     }
26 }
27 v8[15] = ele_15;           // *
28 }
29 else if ( ele_8 == 13 && ele_4 == 37 && v8[0] == 6 )
30 {
31     if ( !ele_13 )
32         v8[16] = 1;         // *
33     if ( !ele_15 )
34         goto LABEL_8;
35 }
36
37 if ( ele_15 == ele_13 )
38     v8[16] = 0;             // *
39
40
41 if ( ele_15 )
42 {
43 LABEL_8:
44     if ( v8[18] && !v8[16] )
45         v8[17] = 1;         // *
46 }
47
48 if ( ele_8 | ele_4 || (result = 1LL, v8[0]) )
49 {
50 LABEL_12:
51     result = 0LL;
52     v8[14] = 0;             // *
53     return result;
54 }
55 v8[14] = 1;                 // *
56 return result;
57 }

```

```

1  int input_basedFunc(__int64 v8, char input_i, _DWORD *len_input)
2  {
3      int v3; // ecx
4      int result; // rax
5      int tmp_o;
6      int tmp_i;
7      int tmp_m; // edx
8
9      v3 = (*len_input)++;
10     if ( input_i == '!' )
11     {
12         (v8[13]) ^= 1u;
13
14     GOOD:

```



```

15     eval_v8Array(v8);
16     result = 0;
17 }
18 else {
19     switch ( input_i )
20     {
21         case '?':
22             suspiciousFun1(v8);
23             --len_input;
24             goto GOOD
25         ;
26
27         case '@':
28             v8[12] ^= 1;
29             *len_input = v3;
30             goto GOOD
31         ;
32         case 'i':case 'I':
33         {
34             if(input_i == 'i')
35                 v8[0] = (v8[0] + 1) % 28;
36             else v8[0] = (v8[0] + 27) %28
37             goto GOOD
38         };
39         case 'm':case 'M':
40         {
41             if(input_i == 'm')
42                 v8[4] = (v8[4] + 1) % 44;
43             else v8[4] = (v8[4] + 43) % 44;
44             goto GOOD
45         };
46         case 'o':case 'O':
47         {
48             if (input_i == 'o')
49                 v8[8] = (v8[8] + 1) % 56;
50             else v8[8] = (v8[8] + 55) %56;
51             goto GOOD
52         };
53         default:
54             result = 1LL;
55             break;
56     }
57 } //kết thúc câu lệnh else
58
59 return result;
60 }

```

Sinh một input thỏa mãn

- Sau khi đã hiểu được hoạt động của file thực thi, mục tiêu của chúng ta là sinh ra input để v14 - v18 phải bằng 1:

```

1 Set theo thứ tự:
2
3 + v8Arr[18] = 1( viết tắt là v18): "IIIIIImmmmmmmmmmmooooooooo"
4     Lúc này v0 = 22, v4 = 11, v8 = 7
5
6 + v16 = 1: "iiiiiiiiiimmmmmmmmmmmmmmmmmmmoooooooo"
7     Lúc này v0 = 6, v4 = 37, v8 = 13
8
9 + "!"
10    input này để v13 = 1
11
12 + "iiimmmmmmmmmmmmmmmmmmmmm000000"
13    Lúc này v0 = 9, v4 = 8, v8 = 7
14
15 + input: "!" để v13 = 0,
16    vì v0, v4, v8 giữ nguyên nên v15 = 1
17
18 + input: "!" sinh ra v13 = 1, và vì v13 = v15 nên v16 = 0,
19    cùng lúc đó v16 = 0 và v18 = 1, nên v17 = 1. Lúc này v0 = 9, v4 = 8, v8
20 + input: "IIIMMMMMMMMMMMMMMMoooooooo!" sinh ra v0 = 6, v4 = 37, v8 = 13
21    v13 = 0 và (v0 = 6, v4 = 37, v8 = 13) nên v16 = 1
22 + "IIIIIImmmmmmmmm0000000000000000"
23    v0 = v4 = v8 = 0 nên v14 = 1
24 ==> như vậy lúc này v14- v18 đã được set bằng 1, sinh ra flag

```

[illegible]

Dysfunctional

Challenge's Description: My "functional programming in C" library finally works! Although, now that I realize it, it seems more... dysfunctional. Either way, I've used it for my [cutting edge flag encryption algorithm](#) along with [this file](#) - here's the [encrypted flag](#).

Nếu không thể truy cập vào file source của server, bạn có thể tìm thấy nó ở đây: [dysfx](#), [not_flag](#), [encrypted_flag](#)

TL; DR

1. Chương trình sẽ đọc byte trong 2 file **flag** và **not_flag**
2. Từng byte trong file **flag** sẽ được nối với 3 bytes sinh ra bởi **/dev/urandom**, như vậy, từ mảng 1-byte flag ban đầu, giờ chúng ta đã có một mảng mới với kích cỡ các phần tử là 4-byte, gọi mảng này là mảng **originalArr**
3. Mỗi phần tử của **originalArr** sẽ gồm 2 word, mỗi word(2 bytes) sẽ được xor với các tham số có sẵn(chẳng hạn 0xdead, 0x1337...), tính toán và sinh ra một index
4. Ánh xạ index này đến mảng byte đọc từ file **not_flag**. Lấy một word tại index đó. Suy ra: từ phần tử 2 word ban đầu –sinh ra–> 2 word mới, và 2 word mới này tạo thành một số trong file **encrypted_flag**:

```
1 5400f172 949c5165 c6bc4607 c621d63f c20c0970 f2b3f53c aabb67f9 fe0e7abb 6a
2
```

6. **Solve:** Từ file **encrypted_flag** suy ra được flag sẽ có 40 ký tự → tách mỗi số thành 2 word, tìm trong file **not_flag** → index. Vì index tạo thành từ các phép tính mang tính **reversible**, vì vậy mà từ mỗi dword được khôi phục lại chúng ta sẽ tách lấy **byte cuối cùng**, chính là một byte của flag

Ngụ ý tên của challenge và viết solve script

Tên Dysfunctional

- Sẽ có 2 hàm rất dễ thấy là **closure()** và **compose()**. Tuy nhiên thực chất, 2 hàm này sẽ được build lại lúc chạy chương trình(**sub_FFD()** sẽ chịu trách nhiệm cho việc tính lại các giá trị cần thiết cho việc build một hàm như **func_start, func_size** v.v.)
 - chức năng thật của **closure()** function: sẽ chịu trách nhiệm cho việc tính toán các phần tử của **originalArr**, tham chiếu đến mảng byte từ file **not_flag**
 - Chú ý, sẽ có 2 bộ tham số khác nhau cho hàm **closure()**: (*0xdead, 0xbeef, 0xfeed*) và (*0x1337, 0xcafe, 0x1234*)
 - 2 lần gọi hàm **compose** thực ra là:
 - compose1** : Gọi hàm **closure()** *theo thứ tự* bộ tham số (*0xdead, 0xbeef, 0xfeed*), tiếp theo là với tham số (*0x1337, 0xcafe, 0x1234*)
 - compose2**: (*0x1337, 0xcafe, 0x1234*), tiếp theo là (*0xdead, 0xbeef, 0xfeed*)
 - map gọi **compose1** và truyền vào mảng **originalArr** để xử lý, sinh ra một mảng mới **resArr1**
 - resArr1** truyền vào map với hàm **compose2**, xử lý và sinh ra các giá trị trong file **encrypted_flag**

```
v5 = closure(0xDEADLL, 0xBEEFLL, 0xFEEDLL, lookup);
v6 = closure(0x1337LL, 0xCAPELL, 0x1234LL, lookup);
compose1 = compose(v5, v6);
compose2 = compose(v6, v5);
resArr1 = map(compose1, originalArr, v15);
encrypted = map(compose2, resArr1, v15);
for ( k = 0; k < (signed int)v15; ++k )
    printf("%x ", *(unsigned int *) (4LL * k + encrypted));
```

Solve Script

Bạn có thể tìm thấy một solve [script](#) của challenge này quanh đây.

