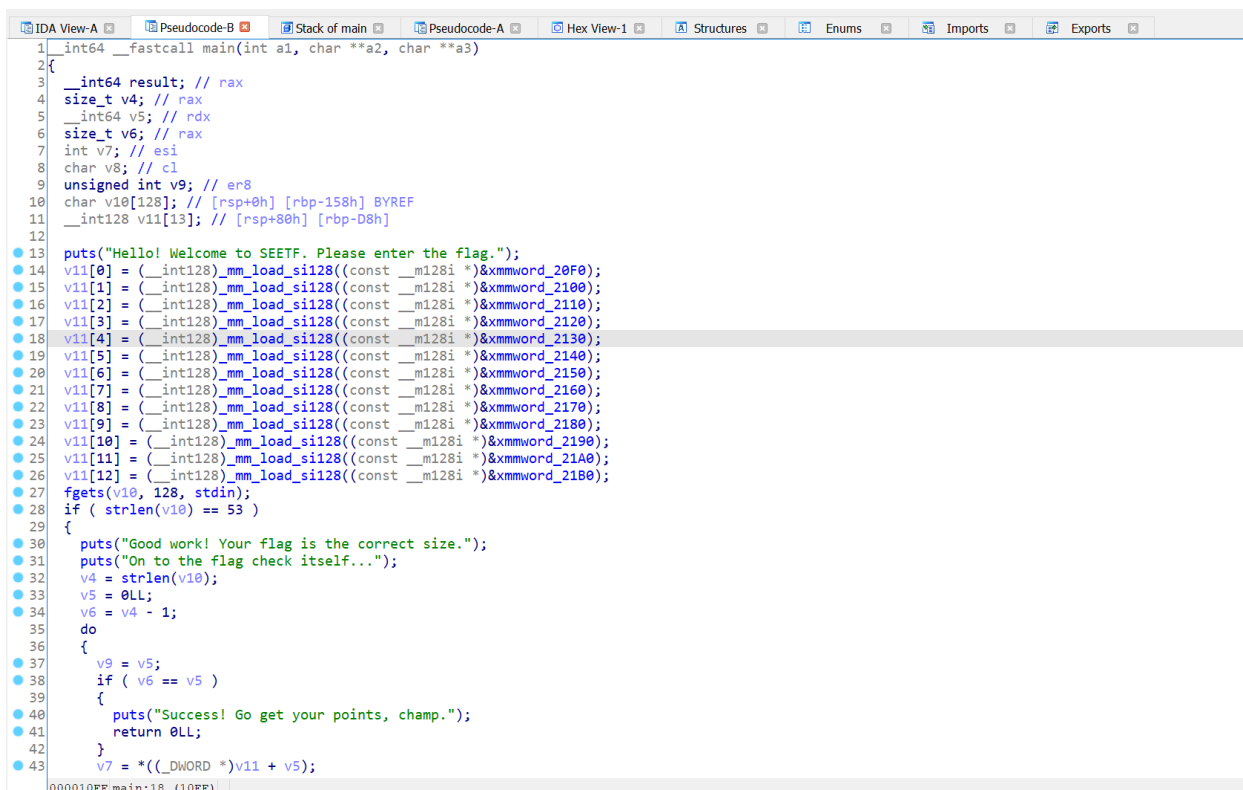


Bài 1:

```
rayinaw@rune:/mnt/hgfs/vmware$ pwn checksec SEETF_chall
[*] '/mnt/hgfs/vmware/SEETF_chall'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       PIE enabled
```

```
rayinaw@rune:/mnt/hgfs/vmware$ file SEETF_chall
SEETF_chall: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=151528987cd274999ec93665ef2d6a7678c5107b, for GNU/Linux 3.2.0, stripped
```

File này là file stripped, nó giảm lược đi những thông tin dư thừa nên chúng ta sẽ không thể tìm thấy hàm main khi debug trong gdb, tuy nhiên với IDA Pro thì ta có thể thấy nó dễ dàng.



```
1  __int64 __fastcall main(int a1, char **a2, char **a3)
2  {
3      __int64 result; // rax
4      size_t v4; // rax
5      __int64 v5; // rdx
6      size_t v6; // rax
7      int v7; // esi
8      char v8; // cl
9      unsigned int v9; // er8
10     char v10[128]; // [rsp+0h] [rbp-158h] BYREF
11     __int128 v11[13]; // [rsp+80h] [rbp-D8h]
12
13     puts("Hello! Welcome to SEETF. Please enter the flag.");
14     v11[0] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_20F0);
15     v11[1] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2100);
16     v11[2] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2110);
17     v11[3] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2120);
18     v11[4] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2130);
19     v11[5] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2140);
20     v11[6] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2150);
21     v11[7] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2160);
22     v11[8] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2170);
23     v11[9] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2180);
24     v11[10] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_2190);
25     v11[11] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_21A0);
26     v11[12] = (__int128)_mm_load_si128((const __m128i *)&__xmmword_21B0);
27     fgets(v10, 128, stdin);
28     if ( strlen(v10) == 53 )
29     {
30         puts("Good work! Your flag is the correct size.");
31         puts("On to the flag check itself...");
32         v4 = strlen(v10);
33         v5 = 0LL;
34         v6 = v4 - 1;
35         do
36         {
37             v9 = v5;
38             if ( v6 == v5 )
39             {
40                 puts("Success! Go get your points, champ.");
41                 return 0LL;
42             }
43             v7 = *((_DWORD *)v11 + v5);
44         } while ( v5++ < v6 );
45     }
46     return 0LL;
47 }
```

Xem một chút về mã giả này:

- Đầu tiên nó lấy đầu vào, rồi kiểm tra độ dài input là v10. Nếu là 53 thì thực hiện tiếp (ở đây nó tính thêm ký tự '\n' nên đầu vào yêu cầu là 52 ký tự), còn không thì in ra "Flag wrong".

```

27 fgets(v10, 128, stdin);
28 if ( strlen(v10) == 53 )
29 {
30     puts("Good work! Your flag is the correct size.");
31     puts("On to the flag check itself...");
32     v4 = strlen(v10);
33     v5 = 0LL;
34     v6 = v4 - 1;
35     do
36     {
37         v9 = v5;
38         if ( v6 == v5 )
39         {
40             puts("Success! Go get your points, champ.");
41             return 0LL;
42         }
43         v7 = *((_DWORD *)v11 + v5);
44         v8 = v5 ^ (v10[v5] + 69);
45         ++v5;
46     }
47     while ( (_BYTE)v7 == v8 );
48     printf("Flag check failed at index: %d", v9);
49     result = 1LL;
50 }
51 else
52 {
53     printf("Flag wrong. Try again.");
54     result = 1LL;
55 }
56 return result;
57 }

```

- Tiếp tục từ dòng 28:
 - + Hàm do while chạy từ v5=0 đến 52 (là len(v10)-1), nếu chạy đến cuối thì puts("Success.....")
 - + Tiếp theo nó tạo v7, v8 từ v11 và v10 rồi so sánh với nhau. Nếu v7==v8 thì chạy tiếp.
- Như kinh nghiệm reverse mình đã từng làm, trước khi xem rõ đoạn tạo và check các ký tự, mình sẽ xem nó liên quan đến hằng nào, có tĩnh hay không.
 - + Ở đây nó liên quan đến v10 là đầu vào ta nhập
 - + Và v11 là một byte string có sẵn, vậy ta chỉ cần tìm được v11 là dễ dàng dịch ngược ra v10.
- Ở đây ta có thể lấy ra v11 từ GDB hoặc Ghidra, và để lấy ra chính xác v11, ta cần set breakpoint ở đoạn nào v11 được tính toán hoàn thành.
- Phần dưới là mình debug bằng GDB nhưng kết quả không đúng, nên nếu muốn đọc để lấy kinh nghiệm debug thì nên đọc, còn k thì quăng xô Ghidra là có ngay.

```

26 v11[12] = (__int128)_mm_10ad_5112b((CONST __int128_t)&XMMWORD_21b);
27 fgets(v10, 128, stdin);
28 if ( strlen(v10) == 53 )
29 {
30     puts("Good work! Your flag is the correct size.");
31     puts("On to the flag check itself...");
32     v4 = strlen(v10);
33     v5 = 0LL;
34     v6 = v4 - 1;
35     do
36     {
37         v9 = v5;
38         if ( v6 == v5 )
39         {
40             puts("Success! Go get your points, champ.");
41             return 0LL;
42         }
43         v7 = *((_DWORD *)v11 + v5);
44         v8 = v5 ^ (v10[v5] + 69);
45         ++v5;
46     }
47     while ( (_BYTE)v7 == v8 );
48     printf("Flag check failed at index: %d", v9);
49     result = 1LL;
50 }
51 else
52 {
53     printf("Flag wrong. Try again.");
54     result = 1LL;
55 }
56 return result;
57 }

```

00001187 main:28 (1187)

Ở đây mình chọn tại vị trí if ở dòng 28. Địa chỉ của nó là 1187 nhưng chưa chính xác đâu nhé.

File này enable PIE nên sẽ làm xáo trộn các địa chỉ tải vào:

```
rayinaw@rune:/mnt/hgfs/vmware$ ldd SEETF_chall
linux-vdso.so.1 (0x00007ffdba7de000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa56c000000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa56c393000)
rayinaw@rune:/mnt/hgfs/vmware$ ldd SEETF_chall
linux-vdso.so.1 (0x00007ffcef71e000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fd2aca00000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd2acd77000)
rayinaw@rune:/mnt/hgfs/vmware$ ldd SEETF_chall
linux-vdso.so.1 (0x00007fffc84ec9000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff413200000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff413603000)
rayinaw@rune:/mnt/hgfs/vmware$ ldd SEETF_chall
linux-vdso.so.1 (0x00007ffffbb5e5000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f179ec00000)
/lib64/ld-linux-x86-64.so.2 (0x00007f179ef07000)
```

Trước khi debug với gdb ta cần dùng lệnh sau: “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space”

Khi debug bằng gdb ta cần run trước rồi làm tiếp để set các địa chỉ mặc định.

Ban đầu khi chưa run, các địa chỉ sẽ như này:

```
pwndbg> info files
Symbols from "/mnt/hgfs/vmware/SEETF_chall".
Local exec file:
  `./mnt/hgfs/vmware/SEETF_chall', file type elf64-x86-64.
Entry point: 0x1220
0x00000000000000318 - 0x00000000000000334 is .interp
0x00000000000000338 - 0x00000000000000358 is .note.gnu.property
0x00000000000000358 - 0x0000000000000037c is .note.gnu.build-id
0x0000000000000037c - 0x0000000000000039c is .note.ABI-tag
0x000000000000003a0 - 0x000000000000003c8 is .gnu.hash
0x000000000000003c8 - 0x000000000000004d0 is .dynsym
0x000000000000004d0 - 0x0000000000000056c is .dynstr
0x0000000000000056c - 0x00000000000000582 is .gnu.version
0x00000000000000588 - 0x000000000000005a8 is .gnu.version_r
0x000000000000005a8 - 0x00000000000000680 is .rela.dyn
0x00000000000000680 - 0x000000000000006e0 is .rela.plt
0x00000000000001000 - 0x00000000000001017 is .init
0x00000000000001020 - 0x00000000000001070 is .plt
0x00000000000001070 - 0x00000000000001078 is .plt.got
0x00000000000001080 - 0x00000000000001381 is .text
0x00000000000001384 - 0x0000000000000138d is .fini
0x00000000000002000 - 0x000000000000021c0 is .rodata
0x000000000000021c0 - 0x00000000000002204 is .eh_frame_hdr
0x00000000000002208 - 0x00000000000002330 is .eh_frame
```



```

pwndbg> x/20i 0x1220
0x1220:    xor     ebp,ebp
0x1222:    mov     r9,rdx
0x1225:    pop     rsi
0x1226:    mov     rdx,rsi
0x1229:    and     rsp,0xfffffffffffffff0
0x122d:    push    rax
0x122e:    push    rsp
0x122f:    lea     r8,[rip+0x14a]          # 0x1380
0x1236:    lea     rcx,[rip+0xe3]          # 0x1320
0x123d:    lea     rdi,[rip+0xffffffffffffe3c] # 0x1080
0x1244:    call    QWORD PTR [rip+0x2d96] # 0x3fe0
0x124a:    hlt
0x124b:    nop     DWORD PTR [rax+rax*1+0x0]
0x1250:    lea     rdi,[rip+0x2df1]          # 0x4048
0x1257:    lea     rax,[rip+0x2dea]          # 0x4048
0x125e:    cmp     rax,rdi
0x1261:    je      0x1278
0x1263:    mov     rax,QWORD PTR [rip+0x2d6e] # 0x3fd8
0x126a:    test    rax,rax
0x126d:    je      0x1278

pwndbg> x/20i 0x00000000000001080
0x1080:    push    rbx
0x1081:    lea     rdi,[rip+0xf80]          # 0x2008
0x1088:    sub     rsp,0x150
0x108f:    call    0x1030 <puts@plt>
0x1094:    movdqa  xmm0,XMMWORD PTR [rip+0x1054] # 0x20f0
0x109c:    mov     rbx,rsp
0x109f:    mov     rdx,QWORD PTR [rip+0x2faa] # 0x4050 <stdin>
0x10a6:    mov     rdi,rbx
0x10a9:    mov     esi,0x80
0x10ae:    movaps  XMMWORD PTR [rsp+0x80],xmm0
0x10b6:    movdqa  xmm0,XMMWORD PTR [rip+0x1042] # 0x2100
0x10be:    movaps  XMMWORD PTR [rsp+0x90],xmm0
0x10c6:    movdqa  xmm0,XMMWORD PTR [rip+0x1042] # 0x2110
0x10ce:    movaps  XMMWORD PTR [rsp+0xa0],xmm0
0x10d6:    movdqa  xmm0,XMMWORD PTR [rip+0x1042] # 0x2120
0x10de:    movaps  XMMWORD PTR [rsp+0xb0],xmm0
0x10e6:    movdqa  xmm0,XMMWORD PTR [rip+0x1042] # 0x2130
0x10ee:    movaps  XMMWORD PTR [rsp+0xc0],xmm0
0x10f6:    movdqa  xmm0,XMMWORD PTR [rip+0x1042] # 0x2140
0x10fe:    movaps  XMMWORD PTR [rsp+0xd0],xmm0

```

Tuy nhiên nếu chúng ta set breakpoint với địa chỉ như thế thì không đúng. Để đúng ta phải run trước rồi kiểm tra lại các địa chỉ:

```

pwndbg> r
Starting program: /mnt/hgfs/vmware/SEETF_chall
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello! Welcome to SEETF. Please enter the flag.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Good work! Your flag is the correct size.
On to the flag check itself...
Flag check failed at index: 0[Inferior 1 (process 4932) exited with code 01]
pwndbg> info files
Symbols from "/mnt/hgfs/vmware/SEETF_chall".
Local exec file:
  '/mnt/hgfs/vmware/SEETF_chall', file type elf64-x86-64.
Entry point: 0x555555555220
0x0000555555554318 - 0x0000555555554334 is .interp
0x0000555555554338 - 0x0000555555554358 is .note.gnu.property
0x0000555555554358 - 0x000055555555437c is .note.gnu.build-id
0x000055555555437c - 0x000055555555439c is .note.ABI-tag
0x00005555555543a0 - 0x00005555555543c8 is .gnu.hash
0x00005555555543c8 - 0x00005555555544d0 is .dynsym
0x00005555555544d0 - 0x000055555555456c is .dynstr
0x000055555555456c - 0x0000555555554582 is .gnu.version
0x0000555555554582 - 0x00005555555545a8 is .gnu.version_r
0x00005555555545a8 - 0x0000555555554680 is .rela.dyn

```

Bây giờ ta lấy địa chỉ hàm main bằng cách:

```

pwndbg> x/20i 0x555555555220
0x555555555220:  xor    ebp,ebp
0x555555555222:  mov    r9,rdx
0x555555555225:  pop    rsi
0x555555555226:  mov    rdx,rsi
0x555555555229:  and    rsp,0xfffffffffffffff0
0x55555555522d:  push   rax
0x55555555522e:  push   rsp
0x55555555522f:  lea    r8,[rip+0x14a]          # 0x555555555380
0x555555555236:  lea    rcx,[rip+0xe3]          # 0x555555555320
0x55555555523d:  lea    rdi,[rip+0xffffffffffffe3c]  # 0x555555555080
0x555555555244:  call   QWORD PTR [rip+0x2d96]     # 0x5555555557fe0
0x55555555524a:  hlt
0x55555555524b:  nop    DWORD PTR [rax+rax*1+0x0]
0x555555555250:  lea    rdi,[rip+0x2df1]          # 0x5555555558048
0x555555555257:  lea    rax,[rip+0x2dea]          # 0x5555555558048
0x55555555525e:  cmp    rax,rdi
0x555555555261:  je     0x555555555278
0x555555555263:  mov    rax,QWORD PTR [rip+0x2d6e]  # 0x5555555557fd8
0x55555555526a:  test   rax,rax
0x55555555526d:  je     0x555555555278

```

Đề ý ở trước lệnh call, nó có thực hiện tính toán địa chỉ, kết quả này là địa chỉ của hàm main "0x555555555080".

Như đã nói ở trên, ta sẽ đặt breakpoint ở đoạn tính toán xong v11 rồi và lấy ra giá trị của v11.

```

pwndbg> x/60i 0x55555555080
0x55555555080:      push    rbx
0x55555555081:      lea     rdi,[rip±0xf80]          # 0x555555556008
0x55555555088:      sub     rsp,0x150
0x5555555508f:      call    0x55555555030 <puts@plt>
0x55555555094:      movdqa  xmm0,XMMWORD PTR [rip±0x1054]      # 0x5555555560f0
0x5555555509c:      mov     rbx,rsp
0x5555555509f:      mov     rdx,QWORD PTR [rip±0x2faa]          # 0x5555555558050 <stdin>
0x555555550a6:      mov     rdi,rbx
0x555555550a9:      mov     esi,0x80
0x555555550ae:      movaps  XMMWORD PTR [rsp±0x80],xmm0
0x555555550b6:      movdqa  xmm0,XMMWORD PTR [rip±0x1042]      # 0x555555556100

```

rayinaw@rune: ~/vmware	×	rayinaw@rune: /mnt/hgfs/vmware	×
0x55555555146:	movdqa	xmm0,XMMWORD PTR [rip±0x1042]	# 0x555555556190
0x5555555514e:	movaps	XMMWORD PTR [rsp±0x120],xmm0	
0x55555555156:	movdqa	xmm0,XMMWORD PTR [rip±0x1042]	# 0x5555555561a0
0x5555555515e:	movaps	XMMWORD PTR [rsp±0x130],xmm0	
0x55555555166:	movdqa	xmm0,XMMWORD PTR [rip±0x1042]	# 0x5555555561b0
0x5555555516e:	movaps	XMMWORD PTR [rsp±0x140],xmm0	
0x55555555176:	call	0x55555555060 <fgets@plt>	
0x5555555517b:	mov	rdi,rbx	
0x5555555517e:	call	0x55555555040 <strlen@plt>	
0x55555555183:	cmp	rax,0x35	
> 0x55555555187:	je	0x555555551a5	
0x55555555189:	lea	rdi,[rip±0xf3c]	# 0x5555555560cc
0x55555555190:	xor	eax,eax	
0x55555555192:	call	0x55555555050 <printf@plt>	
0x55555555197:	mov	eax,0x1	
0x5555555519c:	add	rsp,0x150	
0x555555551a3:	pop	rbx	
0x555555551a4:	ret		
0x555555551a5:	lea	rdi,[rip±0xe8c]	# 0x555555556038
0x555555551ac:	call	0x55555555030 <puts@plt>	
0x555555551b1:	lea	rdi,[rip±0xeb0]	# 0x555555556068

Địa chỉ của hàm if bây giờ là 0x55555555187.

Đặt breakpoint ở 0x55555555187 và run:

```

pwndbg> b*0x55555555187
Breakpoint 3 at 0x55555555187
pwndbg> r
Starting program: /mnt/hgfs/vmware/SEETF_chall
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Hello! Welcome to SEETF. Please enter the flag.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```

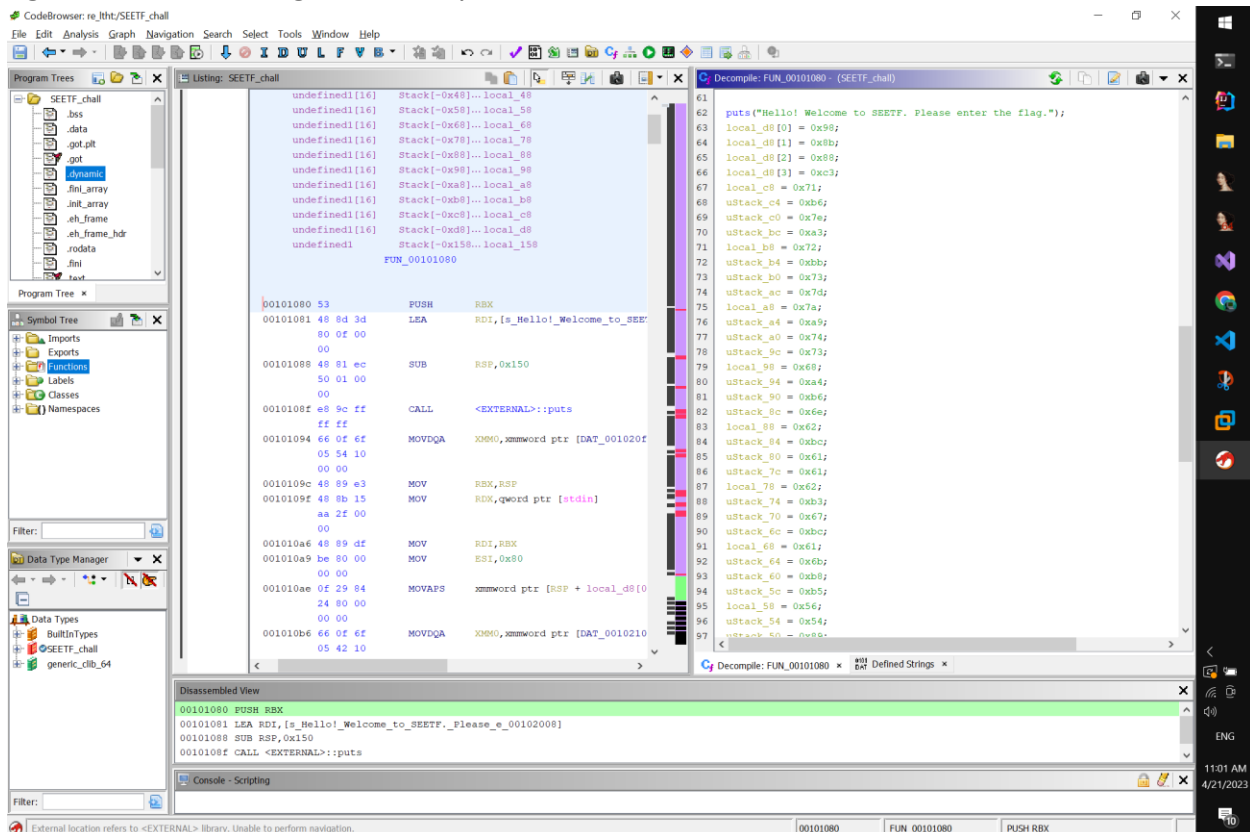
Lấy ra giá trị của mảng byte v11:

```

pwndbg> x/52b $rsp+0x80
0x7fffffffde70: 0x98 0x00 0x00 0x00 0x8b 0x00 0x00 0x00
0x7fffffffde78: 0x88 0x00 0x00 0x00 0xc3 0x00 0x00 0x00
0x7fffffffde80: 0x71 0x00 0x00 0x00 0xb6 0x00 0x00 0x00
0x7fffffffde88: 0x7e 0x00 0x00 0x00 0xa3 0x00 0x00 0x00
0x7fffffffde90: 0x72 0x00 0x00 0x00 0xbb 0x00 0x00 0x00
0x7fffffffde98: 0x73 0x00 0x00 0x00 0x7d 0x00 0x00 0x00
0x7fffffffdea0: 0x7a 0x00 0x00 0x00

```

GDB không ra được byte mong đợi :< bởi vì nhìn là biết nó sẽ tạo ra byte âm nếu dịch ngược, nên thử vào ghidra để lấy xem:



Byte ở đây đã khá rõ ràng, bây giờ ta chỉ cần dịch ngược nó.


```

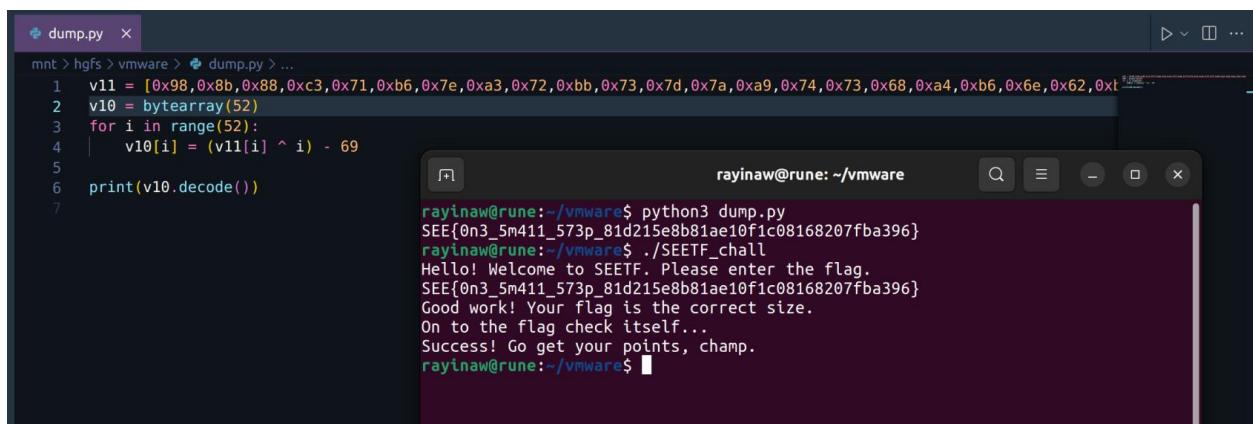
27 fgets(v10, 128, stdin);
28 if ( strlen(v10) == 53 )
29 {
30     puts("Good work! Your flag is the correct size.");
31     puts("On to the flag check itself...");
32     v4 = strlen(v10);
33     v5 = 0LL;
34     v6 = v4 - 1;
35     do
36     {
37         v9 = v5;
38         if ( v6 == v5 )
39         {
40             puts("Success! Go get your points, champ.");
41             return 0LL;
42         }
43         v7 = *((_DWORD *)v11 + v5);
44         v8 = v5 ^ (v10[v5] + 69);
45         ++v5;
46     }
47     while ( (_BYTE)v7 == v8 );
48     printf("Flag check failed at index: %d", v9);

```

Đoạn dịch ngược cũng khá đơn giản:

- Yêu cầu: $v11[i] = i^{(v10[i]+69)}$
- ⇒ $V10[i] = (v11[i]^i)+69$

Kết quả:



```

dump.py x
mnt > hgfs > vmware > dump.py > ...
1  v11 = [0x98,0x8b,0x88,0xc3,0x71,0xb6,0x7e,0xa3,0x72,0xbb,0x73,0x7d,0x7a,0xa9,0x74,0x73,0x68,0xa4,0xb6,0x6e,0x62,0xt...
2  v10 = bytearray(52)
3  for i in range(52):
4      v10[i] = (v11[i] ^ i) - 69
5
6  print(v10.decode())
7
rayinaw@rune: ~/vmware
rayinaw@rune:~/vmware$ python3 dump.py
SEE{0n3_5m411_573p_81d215e8b81ae10f1c08168207fba396}
rayinaw@rune:~/vmware$ ./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.
rayinaw@rune:~/vmware$

```

Bài 2:

Check một chút về file hard_chal:

```
rayinaw@rune:~/vmware$ file hard_chal
hard_chal: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a7b8740f783129c39
91b7efec82bb9bb89e88afb, for GNU/Linux 3.2.0, not stripped
rayinaw@rune:~/vmware$ pwn checksec hard_chal
[*] '/mnt/hgfs/vmware/hard_chal'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

File này not stripped nên chúng ta có thể đọc hàm main bình thường.

```
rayinaw@rune:~/vmware$ ./hard_chal
[$] Enter your input in the form: words_with_underscores_and_letters: test
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: rayinaw
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: abc
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: abc_xyz_gcm
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaa
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaaaaa
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaaa
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaa
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: █
```

Chạy thử chương trình một xíu thì thấy file này sẽ yêu cầu người dùng nhập input cho đến khi nào đúng, nếu sai thì nhập lại. Và chúng ta cần theo đề bài để nhập:

```
[$] Enter your input in the form: words_with_underscores_and_letters: words_with_underscores_and_letters
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaa_aaaa_aaaaaaaaaaaa_aaa_aaaaaaa
[$] Incorrect...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaa_aaaa_aaaaaaaaaaaa_aaa_aaaaaaa
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: aaaaa_aaaa_aaaaaaaaaaaa_aaa_aaaaaaa
[$] This won't do...
[$] Enter your input in the form: words_with_underscores_and_letters: █
```

Để ý nếu ta nhập đúng form là “words_with_underscores_and_letters” thì trả về Incorrect..., còn nếu không đúng form kia thì trả về “This won’t do...”, do gì thì lát nữa tính~.

Dịch ngược file này với IDA để xem rõ nó thực hiện những gì:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     unsigned int v3; // eax
4     unsigned int v4; // eax
5     unsigned int v5; // eax
6     int v6; // eax
7     size_t v7; // rax
8     unsigned __int64 v8; // rax
9     void *v9; // rsp
10    unsigned int v10; // eax
11    unsigned int v11; // eax
12    char v13[15]; // [rsp+8h] [rbp-490h] BYREF
13    char v14; // [rsp+17h] [rbp-481h]
14    unsigned int v15; // [rsp+18h] [rbp-480h]
15    unsigned int i; // [rsp+1Ch] [rbp-47Ch]
16    unsigned int j; // [rsp+20h] [rbp-478h]
17    unsigned int k; // [rsp+24h] [rbp-474h]
18    size_t v19; // [rsp+28h] [rbp-470h]
19    char *dest; // [rsp+30h] [rbp-468h]
20    char v21[64]; // [rsp+38h] [rbp-460h] BYREF
21    char s[1000]; // [rsp+78h] [rbp-420h] BYREF
22    unsigned __int64 v23; // [rsp+460h] [rbp-38h]
23
24    v23 = __readfsqword(0x28u);
25    v14 = 1;
26    while ( v14 )
27    {
28        printf("[%s] Enter your input in the form: words_with_underscores_and_letters: ");
29        __isoc99_scanf("%s", s);
30        v15 = 0;
31        for ( i = 0; ; i = ::s(i) )
32        {
33            v3 = strlen(s);
34            if ( !(unsigned __int8)l(i, v3) )
35                break;
36            if ( (unsigned __int8)eq((unsigned int)s[i], 95LL) != 1 )
37                v15 = ::s(v15);
38        }
39        v4 = strlen(s);
40        if ( (unsigned __int8)ev(v4) != 1
41            || (v5 = strlen(s), v6 = su(v5, 1LL), (unsigned __int8)eq((unsigned int)s[v6], 95LL))
42            || (unsigned __int8)v(s) != 1
43            || (unsigned __int8)ev(v15) != 1 )
44            ;
45    }
46    ;
47    ;
48    ;
49    ;
50    ;
51    ;
52    ;
53    ;
54    ;
55    ;
56    ;
57    ;
58    ;
59    ;
60    ;
61    ;
62    ;
63    ;
64    ;
65    ;
66    ;
67    ;
68    ;
69    ;
70    ;
71    ;
72    ;
73    ;
74    ;
75    ;
76    ;
77    ;
78    ;
79    ;
80    ;
81    ;
82    ;
83    ;
84    ;
85    ;
86    ;
87    ;
88    ;
89    ;
90    ;
91    ;
92    ;
93    ;
94    ;
95    ;
96    ;
97    ;
98    ;
99    ;
100   ;
101   ;
102   ;
103   ;
104   ;
105   ;
106   ;
107   ;
108   ;
109   ;
110   ;
111   ;
112   ;
113   ;
114   ;
115   ;
116   ;
117   ;
118   ;
119   ;
120   ;
121   ;
122   ;
123   ;
124   ;
125   ;
126   ;
127   ;
128   ;
129   ;
130   ;
131   ;
132   ;
133   ;
134   ;
135   ;
136   ;
137   ;
138   ;
139   ;
140   ;
141   ;
142   ;
143   ;
144   ;
145   ;
146   ;
147   ;
148   ;
149   ;
150   ;
151   ;
152   ;
153   ;
154   ;
155   ;
156   ;
157   ;
158   ;
159   ;
160   ;
161   ;
162   ;
163   ;
164   ;
165   ;
166   ;
167   ;
168   ;
169   ;
170   ;
171   ;
172   ;
173   ;
174   ;
175   ;
176   ;
177   ;
178   ;
179   ;
180   ;
181   ;
182   ;
183   ;
184   ;
185   ;
186   ;
187   ;
188   ;
189   ;
190   ;
191   ;
192   ;
193   ;
194   ;
195   ;
196   ;
197   ;
198   ;
199   ;
200   ;
201   ;
202   ;
203   ;
204   ;
205   ;
206   ;
207   ;
208   ;
209   ;
210   ;
211   ;
212   ;
213   ;
214   ;
215   ;
216   ;
217   ;
218   ;
219   ;
220   ;
221   ;
222   ;
223   ;
224   ;
225   ;
226   ;
227   ;
228   ;
229   ;
230   ;
231   ;
232   ;
233   ;
234   ;
235   ;
236   ;
237   ;
238   ;
239   ;
240   ;
241   ;
242   ;
243   ;
244   ;
245   ;
246   ;
247   ;
248   ;
249   ;
250   ;
251   ;
252   ;
253   ;
254   ;
255   ;
256   ;
257   ;
258   ;
259   ;
260   ;
261   ;
262   ;
263   ;
264   ;
265   ;
266   ;
267   ;
268   ;
269   ;
270   ;
271   ;
272   ;
273   ;
274   ;
275   ;
276   ;
277   ;
278   ;
279   ;
280   ;
281   ;
282   ;
283   ;
284   ;
285   ;
286   ;
287   ;
288   ;
289   ;
290   ;
291   ;
292   ;
293   ;
294   ;
295   ;
296   ;
297   ;
298   ;
299   ;
300   ;
301   ;
302   ;
303   ;
304   ;
305   ;
306   ;
307   ;
308   ;
309   ;
310   ;
311   ;
312   ;
313   ;
314   ;
315   ;
316   ;
317   ;
318   ;
319   ;
320   ;
321   ;
322   ;
323   ;
324   ;
325   ;
326   ;
327   ;
328   ;
329   ;
330   ;
331   ;
332   ;
333   ;
334   ;
335   ;
336   ;
337   ;
338   ;
339   ;
340   ;
341   ;
342   ;
343   ;
344   ;
345   ;
346   ;
347   ;
348   ;
349   ;
350   ;
351   ;
352   ;
353   ;
354   ;
355   ;
356   ;
357   ;
358   ;
359   ;
360   ;
361   ;
362   ;
363   ;
364   ;
365   ;
366   ;
367   ;
368   ;
369   ;
370   ;
371   ;
372   ;
373   ;
374   ;
375   ;
376   ;
377   ;
378   ;
379   ;
380   ;
381   ;
382   ;
383   ;
384   ;
385   ;
386   ;
387   ;
388   ;
389   ;
390   ;
391   ;
392   ;
393   ;
394   ;
395   ;
396   ;
397   ;
398   ;
399   ;
400   ;
401   ;
402   ;
403   ;
404   ;
405   ;
406   ;
407   ;
408   ;
409   ;
410   ;
411   ;
412   ;
413   ;
414   ;
415   ;
416   ;
417   ;
418   ;
419   ;
420   ;
421   ;
422   ;
423   ;
424   ;
425   ;
426   ;
427   ;
428   ;
429   ;
430   ;
431   ;
432   ;
433   ;
434   ;
435   ;
436   ;
437   ;
438   ;
439   ;
440   ;
441   ;
442   ;
443   ;
444   ;
445   ;
446   ;
447   ;
448   ;
449   ;
450   ;
451   ;
452   ;
453   ;
454   ;
455   ;
456   ;
457   ;
458   ;
459   ;
460   ;
461   ;
462   ;
463   ;
464   ;
465   ;
466   ;
467   ;
468   ;
469   ;
470   ;
471   ;
472   ;
473   ;
474   ;
475   ;
476   ;
477   ;
478   ;
479   ;
480   ;
481   ;
482   ;
483   ;
484   ;
485   ;
486   ;
487   ;
488   ;
489   ;
490   ;
491   ;
492   ;
493   ;
494   ;
495   ;
496   ;
497   ;
498   ;
499   ;
500   ;
501   ;
502   ;
503   ;
504   ;
505   ;
506   ;
507   ;
508   ;
509   ;
510   ;
511   ;
512   ;
513   ;
514   ;
515   ;
516   ;
517   ;
518   ;
519   ;
520   ;
521   ;
522   ;
523   ;
524   ;
525   ;
526   ;
527   ;
528   ;
529   ;
530   ;
531   ;
532   ;
533   ;
534   ;
535   ;
536   ;
537   ;
538   ;
539   ;
540   ;
541   ;
542   ;
543   ;
544   ;
545   ;
546   ;
547   ;
548   ;
549   ;
550   ;
551   ;
552   ;
553   ;
554   ;
555   ;
556   ;
557   ;
558   ;
559   ;
560   ;
561   ;
562   ;
563   ;
564   ;
565   ;
566   ;
567   ;
568   ;
569   ;
570   ;
571   ;
572   ;
573   ;
574   ;
575   ;
576   ;
577   ;
578   ;
579   ;
580   ;
581   ;
582   ;
583   ;
584   ;
585   ;
586   ;
587   ;
588   ;
589   ;
590   ;
591   ;
592   ;
593   ;
594   ;
595   ;
596   ;
597   ;
598   ;
599   ;
600   ;
601   ;
602   ;
603   ;
604   ;
605   ;
606   ;
607   ;
608   ;
609   ;
610   ;
611   ;
612   ;
613   ;
614   ;
615   ;
616   ;
617   ;
618   ;
619   ;
620   ;
621   ;
622   ;
623   ;
624   ;
625   ;
626   ;
627   ;
628   ;
629   ;
630   ;
631   ;
632   ;
633   ;
634   ;
635   ;
636   ;
637   ;
638   ;
639   ;
640   ;
641   ;
642   ;
643   ;
644   ;
645   ;
646   ;
647   ;
648   ;
649   ;
650   ;
651   ;
652   ;
653   ;
654   ;
655   ;
656   ;
657   ;
658   ;
659   ;
660   ;
661   ;
662   ;
663   ;
664   ;
665   ;
666   ;
667   ;
668   ;
669   ;
670   ;
671   ;
672   ;
673   ;
674   ;
675   ;
676   ;
677   ;
678   ;
679   ;
680   ;
681   ;
682   ;
683   ;
684   ;
685   ;
686   ;
687   ;
688   ;
689   ;
690   ;
691   ;
692   ;
693   ;
694   ;
695   ;
696   ;
697   ;
698   ;
699   ;
700   ;
701   ;
702   ;
703   ;
704   ;
705   ;
706   ;
707   ;
708   ;
709   ;
710   ;
711   ;
712   ;
713   ;
714   ;
715   ;
716   ;
717   ;
718   ;
719   ;
720   ;
721   ;
722   ;
723   ;
724   ;
725   ;
726   ;
727   ;
728   ;
729   ;
730   ;
731   ;
732   ;
733   ;
734   ;
735   ;
736   ;
737   ;
738   ;
739   ;
740   ;
741   ;
742   ;
743   ;
744   ;
745   ;
746   ;
747   ;
748   ;
749   ;
750   ;
751   ;
752   ;
753   ;
754   ;
755   ;
756   ;
757   ;
758   ;
759   ;
760   ;
761   ;
762   ;
763   ;
764   ;
765   ;
766   ;
767   ;
768   ;
769   ;
770   ;
771   ;
772   ;
773   ;
774   ;
775   ;
776   ;
777   ;
778   ;
779   ;
780   ;
781   ;
782   ;
783   ;
784   ;
785   ;
786   ;
787   ;
788   ;
789   ;
790   ;
791   ;
792   ;
793   ;
794   ;
795   ;
796   ;
797   ;
798   ;
799   ;
800   ;
801   ;
802   ;
803   ;
804   ;
805   ;
806   ;
807   ;
808   ;
809   ;
810   ;
811   ;
812   ;
813   ;
814   ;
815   ;
816   ;
817   ;
818   ;
819   ;
820   ;
821   ;
822   ;
823   ;
824   ;
825   ;
826   ;
827   ;
828   ;
829   ;
830   ;
831   ;
832   ;
833   ;
834   ;
835   ;
836   ;
837   ;
838   ;
839   ;
840   ;
841   ;
842   ;
843   ;
844   ;
845   ;
846   ;
847   ;
848   ;
849   ;
850   ;
851   ;
852   ;
853   ;
854   ;
855   ;
856   ;
857   ;
858   ;
859   ;
860   ;
861   ;
862   ;
863   ;
864   ;
865   ;
866   ;
867   ;
868   ;
869   ;
870   ;
871   ;
872   ;
873   ;
874   ;
875   ;
876   ;
877   ;
878   ;
879   ;
880   ;
881   ;
882   ;
883   ;
884   ;
885   ;
886   ;
887   ;
888   ;
889   ;
890   ;
891   ;
892   ;
893   ;
894   ;
895   ;
896   ;
897   ;
898   ;
899   ;
900   ;
901   ;
902   ;
903   ;
904   ;
905   ;
906   ;
907   ;
908   ;
909   ;
910   ;
911   ;
912   ;
913   ;
914   ;
915   ;
916   ;
917   ;
918   ;
919   ;
920   ;
921   ;
922   ;
923   ;
924   ;
925   ;
926   ;
927   ;
928   ;
929   ;
930   ;
931   ;
932   ;
933   ;
934   ;
935   ;
936   ;
937   ;
938   ;
939   ;
940   ;
941   ;
942   ;
943   ;
944   ;
945   ;
946   ;
947   ;
948   ;
949   ;
950   ;
951   ;
952   ;
953   ;
954   ;
955   ;
956   ;
957   ;
958   ;
959   ;
960   ;
961   ;
962   ;
963   ;
964   ;
965   ;
966   ;
967   ;
968   ;
969   ;
970   ;
971   ;
972   ;
973   ;
974   ;
975   ;
976   ;
977   ;
978   ;
979   ;
980   ;
981   ;
982   ;
983   ;
984   ;
985   ;
986   ;
987   ;
988   ;
989   ;
990   ;
991   ;
992   ;
993   ;
994   ;
995   ;
996   ;
997   ;
998   ;
999   ;
1000  ;
1001  ;
1002  ;
1003  ;
1004  ;
1005  ;
1006  ;
1007  ;
1008  ;
1009  ;
1010  ;
1011  ;
1012  ;
1013  ;
1014  ;
1015  ;
1016  ;
1017  ;
1018  ;
1019  ;
1020  ;
1021  ;
1022  ;
1023  ;
1024  ;
1025  ;
1026  ;
1027  ;
1028  ;
1029  ;
1030  ;
1031  ;
1032  ;
1033  ;
1034  ;
1035  ;
1036  ;
1037  ;
1038  ;
1039  ;
1040  ;
1041  ;
1042  ;
1043  ;
1044  ;
1045  ;
1046  ;
1047  ;
1048  ;
1049  ;
1050  ;
1051  ;
1052  ;
1053  ;
1054  ;
1055  ;
1056  ;
1057  ;
1058  ;
1059  ;
1060  ;
1061  ;
1062  ;
1063  ;
1064  ;
1065  ;
1066  ;
1067  ;
1068  ;
1069  ;
1070  ;
1071  ;
1072  ;
1073  ;
1074  ;
1075  ;
1076  ;
1077  ;
1078  ;
1079  ;
1080  ;
1081  ;
1082  ;
1083  ;
1084  ;
1085  ;
1086  ;
1087  ;
1088  ;
1089  ;
1090  ;
1091  ;
1092  ;
1093  ;
1094  ;
1095  ;
1096  ;
1097  ;
1098  ;
1099  ;
1100  ;
1101  ;
1102  ;
1103  ;
1104  ;
1105  ;
1106  ;
1107  ;
1108  ;
1109  ;
1110  ;
1111  ;
1112  ;
1113  ;
1114  ;
1115  ;
1116  ;
1117  ;
1118  ;
1119  ;
1120  ;
1121  ;
1122  ;
1123  ;
1124  ;
1125  ;
1126  ;
1127  ;
1128  ;
1129  ;
1130  ;
1131  ;
1132  ;
1133  ;
1134  ;
1135  ;
1136  ;
1137  ;
1138  ;
1139  ;
1140  ;
1141  ;
1142  ;
1143  ;
1144  ;
1145  ;
1146  ;
1147  ;
1148  ;
1149  ;
1150  ;
1151  ;
1152  ;
1153  ;
1154  ;
1155  ;
1156  ;
1157  ;
1158  ;
1159  ;
1160  ;
1161  ;
1162  ;
1163  ;
1164  ;
1165  ;
1166  ;
1167  ;
1168  ;
1169  ;
1170  ;
1171  ;
1172  ;
1173  ;
1174  ;
1175  ;
1176  ;
1177  ;
1178  ;
1179  ;
1180  ;
1181  ;
1182  ;
1183  ;
1184  ;
1185  ;
1186  ;
1187  ;
1188  ;
1189  ;
1190  ;
1191  ;
1192  ;
1193  ;
1194  ;
1195  ;
1196  ;
1197  ;
1198  ;
1199  ;
1200  ;
1201  ;
1202  ;
1203  ;
1204  ;
1205  ;
1206  ;
1207  ;
1208  ;
1209  ;
1210  ;
1211  ;
1212  ;
1213  ;
1214  ;
1215  ;
1216  ;
1217  ;
1218  ;
1219  ;
1220  ;
1221  ;
1222  ;
1223  ;
1224  ;
1225  ;
1226  ;
1227  ;
1228  ;
1229  ;
1230  ;
1231  ;
1232  ;
1233  ;
1234  ;
1235  ;
1236  ;
1237  ;
1238  ;
1239  ;
1240  ;
1241  ;
1242  ;
1243  ;
1244  ;
1245  ;
1246  ;
1247  ;
1248  ;
1249  ;
1250  ;
1251  ;
1252  ;
1253  ;
1254  ;
1255  ;
1256  ;
1257  ;
1258  ;
1259  ;
1260  ;
1261  ;
1262  ;
1263  ;
1264  ;
1265  ;
1266  ;
1267  ;
1268  ;
1269  ;
1270  ;
1271  ;
1272  ;
1273  ;
1274  ;
1275  ;
1276  ;
1277  ;
1278  ;
1279  ;
1280  ;
1281  ;
1282  ;
1283  ;
1284  ;
1285  ;
1286  ;
1287  ;
1288  ;
1289  ;
1290  ;
1291  ;
1292  ;
1293  ;
1294  ;
1295  ;
1296  ;
1297  ;
1298  ;
1299  ;
1300  ;
1301  ;
1302  ;
1303  ;
1304  ;
1305  ;
1306  ;
1307  ;
1308  ;
1309  ;
1310  ;
1311  ;
1312  ;
1313  ;
1314  ;
1315  ;
1316  ;
1317  ;
1318  ;
1319  ;
1320  ;
1321  ;
1322  ;
1323  ;
1324  ;
1325  ;
1326  ;
1327  ;
1328  ;
1329  ;
1330  ;
1331  ;
1332  ;
1333  ;
1334  ;
1335  ;
1336  ;
1337  ;
1338  ;
1339  ;
1340  ;
1341  ;
1342  ;
1343  ;
1344  ;
1345  ;
1346  ;
1347  ;
1348  ;
1349  ;
1350  ;
1351  ;
1352  ;
1353  ;
1354  ;
1355  ;
1356  ;
1357  ;
1358  ;
1359  ;
1360  ;
1361  ;
1362  ;
1363  ;
1364  ;
1365  ;
1366  ;
1367  ;
1368  ;
1369  ;
1370  ;
1371  ;
1372  ;
1373  ;
1374  ;
1375  ;
1376  ;
1377  ;
1378  ;
1379  ;
1380  ;
1381  ;
1382  ;
1383  ;
1384  ;
1385  ;
1386  ;
1387  ;
1388  ;
1389  ;
1390  ;
1391  ;
1392  ;
1393  ;
1394  ;
1395  ;
1396  ;
1397  ;
1398  ;
1399  ;
1400  ;
1401  ;
1402  ;
1403  ;
1404  ;
1405  ;
1406  ;
1407  ;
1408  ;
1409  ;
1410  ;
1411  ;
1412  ;
1413  ;
1414  ;
1415  ;
1416  ;
1417  ;
1418  ;
1419  ;
1420  ;
1421  ;
1422  ;
1423  ;
1424  ;
1425  ;
1426  ;
1427  ;
1428  ;
1429  ;
1430  ;
1431  ;
1432  ;
1433  ;
1434  ;
1435  ;
1436  ;
1437  ;
1438  ;
1439  ;
1440  ;
1441  ;
1442  ;
1443  ;
1444  ;
1445  ;
1446  ;
1447  ;
1448  ;
1449  ;
1450  ;
1451  ;
1452  ;
1453  ;
1454  ;
1455  ;
1456  ;
1457  ;
1458  ;
1459  ;
1460  ;
1461  ;
1462  ;
1463  ;
1464  ;
1465  ;
1466  ;
1467  ;
1468  ;
1469  ;
1470  ;
1471  ;
1472  ;
1473  ;
1474  ;
1475  ;
1476  ;
1477  ;
1478  ;
1479  ;
1480  ;
1481  ;
1482  ;
1483  ;
1484  ;
1485  ;
1486  ;
1487  ;
1488  ;
1489  ;
1490  ;
1491  ;
1492  ;
1493  ;
1494  ;
1495  ;
1496  ;
1497  ;
1498  ;
1499  ;
1500  ;
1501  ;
1502  ;
1503  ;
1504  ;
1505  ;
1506  ;
1507  ;
1508  ;
1509  ;
1510  ;
1511  ;
1512  ;
1513  ;
1514  ;
1515  ;
1516  ;
1517  ;
1518  ;
1519  ;
1520  ;
1521  ;
1522  ;
1523  ;
1524  ;
1525  ;
1526  ;
1527  ;
1528  ;
1529  ;
1530  ;
1531  ;
1532  ;
1533  ;
1534  ;
1535  ;
1536  ;
1537  ;
1538  ;
1539  ;
1540  ;
1541  ;
1542  ;
1543  ;
1544  ;
1545  ;
1546  ;
1547  ;
1548  ;
1549  ;
1550  ;
1551  ;
1552  ;
1553  ;
1554  ;
1555  ;
1556  ;
1557  ;
1558  ;
1559  ;
1560  ;
1561  ;
1562  ;
1563  ;
1564  ;
1565  ;
1566  ;
1567  ;
1568  ;
1569  ;
1570  ;
1571  ;
1572  ;
1573  ;
1574  ;
1575  ;
1576  ;
1577  ;
1578  ;
1579  ;
1580  ;
1581  ;
1582  ;
1583  ;
1584  ;
1585  ;
1586  ;
1587  ;
1588  ;
1589  ;
1590  ;
1591  ;
1592  ;
1593  ;
1594  ;
1595  ;
1596  ;
1597  ;
1598  ;
1599  ;
1600  ;
1601  ;
1602  ;
1603  ;
1604  ;
1605  ;
1606  ;
1607  ;
1608  ;
1609  ;
1610  ;
1611  ;
1612  ;
1613  ;
1614  ;
1615  ;
1616  ;
1617  ;
1618  ;
1619  ;
1620  ;
1621  ;
1622  ;
1623  ;
1624  ;
1625  ;
1626  ;
1627  ;
1628  ;
1629  ;
1630  ;
1631  ;
1632  ;
1633  ;
1634  ;
1635  ;
1636  ;
1637  ;
1638  ;
1639  ;
1640  ;
1641  ;
1642  ;
1643  ;
1644  ;
1645  ;
1646  ;
1647  ;
1648  ;
1649  ;
1650  ;
1651  ;
1652  ;
1653  ;
1654  ;
1655  ;
1656  ;
1657  ;
1658  ;
1659  ;
1660  ;
1661  ;
1662  ;
1663  ;
1664  ;
1665  ;
1666  ;
1667  ;
1668  ;
1669  ;
1670  ;
1671  ;
1672  ;
1673  ;
1674  ;
1675  ;
1676  ;
1677  ;
1678  ;
1679  ;
1680  ;
1681  ;
1682  ;
1683  ;
1684  ;
1685  ;
1686  ;
1687  ;
1688  ;
1689  ;
1690  ;
1691  ;
1692  ;
1693  ;
1694  ;
1695  ;
1696  ;
1697  ;
1698  ;
1699  ;
1700  ;
1701  ;
1702  ;
1703  ;
1704  ;
1705  ;
1706  ;
1707  ;
1708  ;
1709  ;
1710  ;
1711  ;
1712  ;
1713  ;
1714  ;
1715  ;
1716  ;
1717  ;
1718  ;
1719  ;
1720  ;
1721  ;
1722  ;
1723  ;
1724  ;
1725  ;
1726  ;
1727  ;
1728  ;
1729  ;
1730  ;
1731  ;
1732  ;
1733  ;
1734  ;
1735  ;
1736  ;
1737  ;
1738  ;
1739  ;
1740  ;
1741  ;
1742  ;
1743  ;
1744  ;
1745  ;
1746  ;
1747  ;
1748  ;
1749  ;
1750  ;
1751  ;
1752  ;
1753  ;
1754  ;
1755  ;
1756  ;
1757  ;
1758  ;
1759  ;
1760  ;
1761  ;
1762  ;
1763  ;
1764  ;
1765  ;
1766  ;
1767  ;
1768  ;
1769  ;
1770  ;
1771  ;
1772  ;
1773  ;
1774  ;
1775  ;
1776  ;
1777  ;
1778  ;
1779  ;
1780  ;
1781  ;
1782  ;
1783  ;
1784  ;
1785  ;
1786  ;
1787  ;
1788  ;
1789  ;
1790  ;
1791  ;
1792  ;
1793  ;
1794  ;
1795  ;
1796  ;
1797  ;
1798  ;
1799  ;
1800  ;
1801  ;
1802  ;
1803  ;
1804  ;
1805  ;
1806  ;
1807  ;
1808  ;
1809  ;
1810  ;
1811  ;
1812  ;
1813  ;
1814  ;
1815  ;
1816  ;
1817  ;
1818  ;
1819  ;
1820  ;
1821  ;
1822  ;
1823  ;
1824  ;
1825  ;
1826  ;
1827  ;
1828  ;
1829  ;
1830  ;
1831  ;
1832  ;
1833  ;
1834  ;
1835  ;
1836  ;
1837  ;
1838  ;
1839  ;
1840  ;
1841  ;

```

```

42     || (unsigned __int8)v(s) != 1
43     || (unsigned __int8)ev(v15) != 1 )
44 {
45     puts("[<strong>$</strong>] This won't do...");
46 }
47 else
48 {
49     v7 = strlen(s);
50     v19 = v7 - 1;
51     v8 = 16 * ((v7 + 15) / 0x10);
52     while ( v13 != &v13[-(v8 & 0xFFFFFFFFFFFFFFFF000LL)] )
53     ;
54     v9 = alloca(v8 & 0xFFF);
55     if ( (v8 & 0xFFF) != 0 )
56         *(_QWORD *)&v13[(v8 & 0xFFF) - 8] = *(_QWORD *)&v13[(v8 & 0xFFF) - 8];
57     dest = v13;
58     strcpy(v13, s);
59     for ( j = 0; ; j = encode(dest, j) )
60     {
61         v10 = strlen(dest);
62         if ( !(unsigned __int8)l(j, v10) )
63             break;
64     }
65     v11 = strcmp(dest, "odt_sjtfnb_jc_c_fiajb_he_cihh_nkn_atvfjp");
66     if ( (unsigned __int8)eq(v11, 0LL) )
67     {
68         puts("[<strong>$</strong>] Correct!");
69         v14 = 0;
70     }
71     else
72     {
73         puts("[<strong>$</strong>] Incorrect...");
74     }
75 }
76 }
77 qmemcpy(v21, "uiuctf{", 7);
78 for ( k = 0; (unsigned __int8)l(k, 40LL); k = ::s(k) )
79     v21[k + 7] = s[k];
80 v21[47] = 125;
81 v21[48] = 0;
82 printf("[<strong>$</strong>] %s\n", v21);
83 return 0;
84 }

```

Đầu tiên là có một vòng lặp while để check input như này chúng ta test (việc test thử chương trình giúp đọc code nhanh hơn rất nhiều) . Nếu như thoát ra khỏi vòng while này với các điều kiện check đúng thì nó sẽ in ra flag là biến v21, ta sẽ quan tâm nó sau.

Bây giờ ta cùng đi vào đoạn check trong vòng lặp:

- Ta thấy đoạn trên check một cái gì đó, và một là nó in ra “this won’t do”, hai là nó thực hiện phần else để in ra “Correct” hoặc “Incorrect...”


```

30 v15 = 0;
31 for ( i = 0; ; i = ::s(i) )
32 {
33     v3 = strlen(s);
34     if ( !(unsigned __int8)l(i, v3) )
35         break;
36     if ( (unsigned __int8)eq((unsigned int)s[i], 95LL) != 1 )
37         v15 = ::s(v15);
38 }
39 v4 = strlen(s);
40 if ( (unsigned __int8)ev(v4) != 1
41     || (v5 = strlen(s), v6 = su(v5, 1LL), (unsigned __int8)eq((unsigned int)s[v6], 95LL))
42     || (unsigned __int8)v(s) != 1
43     || (unsigned __int8)ev(v15) != 1 )
44 {
45     puts("$ This won't do...");
46 }
47 else
48 {
49     v7 = strlen(s);
50     v19 = v7 - 1;
51     v8 = 16 * ((v7 + 15) / 0x10);
52     while ( v13 != &v13[-(v8 & 0xFFFFFFFFFFFF000LL)] )
53         ;

```

- Vậy thực chất đoạn code trên chỉ là check nó đúng form, nếu đúng thì thực hiện phần trong else, còn không đúng form kiểu words_with_underscores_and_letters

Bây giờ ta cùng check tiếp phần check trong else:

```

47 else
48 {
49     v7 = strlen(s);
50     v19 = v7 - 1;
51     v8 = 16 * ((v7 + 15) / 0x10);
52     while ( v13 != &v13[-(v8 & 0xFFFFFFFFFFFF000LL)] )
53         ;
54     v9 = alloca(v8 & 0xFFF);
55     if ( (v8 & 0xFFF) != 0 )
56         *(_QWORD *)&v13[(v8 & 0xFFF) - 8] = *(_QWORD *)&v13[(v8 & 0xFFF) - 8];
57     dest = v13;
58     strcpy(v13, s);
59     for ( j = 0; ; j = encode(dest, j) )
60     {
61         v10 = strlen(dest);
62         if ( !(unsigned __int8)l(j, v10) )
63             break;
64     }
65     v11 = strcmp(dest, "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp");
66     if ( (unsigned __int8)eq(v11, 0LL) )
67     {
68         puts("$ Correct!");
69         v14 = 0;
70     }
71     else
72     {
73         puts("$ Incorrect...");
74     }
75 }
76 }
77 memcpy(v21, "uiuctf{", 7);
78 for ( k = 0; (unsigned __int8)l(k, 40LL); k = ::s(k) )
79     v21[k + 7] = s[k];
80 v21[47] = 125;

```

Ta cùng đọc từ dưới lên để hiểu cách nó hoạt động:

- Nó so sánh dest với "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp".
- Eq ở đây là hàm so sánh bằng nhau giữa hai tham số:

```
1 int64 __fastcall eq(int a1, int a2)
2 {
3     if ( !a1 && !a2 )
4         return 1LL;
5     if ( a1 && a2 )
6         return eq((unsigned int)(a1 - 1), (unsigned int)(a2 - 1));
7     return 0LL;
8 }
```

- Vậy nếu dest bằng chuỗi kia thì trả về correct, thoát khỏi while và in ra flag.
- Tiếp theo:

```
57 dest = v13;
58 strcpy(v13, s);
59 for ( j = 0; ; j = encode(dest, j) )
60 {
61     v10 = strlen(dest);
62     if ( !(unsigned __int8)l(j, v10) )
63         break;
64 }
65 v11 = strcmp(dest, "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp");
```

Dest ở đây được gán bằng v13 và thực hiện một cái gì đó. Ta cùng làm rõ nó nào:

- + Dest = v13 chỉ định dest trỏ đến vùng nhớ của v13.
- + Sau đó nó thực hiện strcpy string s ta nhập vào v13. Có nghĩa ở đây là dest sẽ gán bằng s.
- + Sau vòng for trên, nếu dest bằng "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp" thì input nhập vào là đúng.

+ Đọc tới đây ta sẽ thấy một điều:

```
49 v7 = strlen(s);
50 v19 = v7 - 1;
51 v8 = 16 * ((v7 + 15) / 0x10);
52 while ( v13 != &v13[-(v8 & 0xFFFFFFFFFFFF0000LL)] )
53     ;
54 v9 = alloca(v8 & 0xFFF);
55 if ( (v8 & 0xFFF) != 0 )
56     *(_QWORD *)&v13[(v8 & 0xFFF) - 8] = *(_QWORD *)&v13[(v8 & 0xFFF) - 8];
57 dest = v13;
58 strcpy(v13, s);
```

Để ý đoạn trên nó không động đến s mà chỉ động đến strlen(s), vậy ta có thể hiểu một cách đơn giản ở đây nó cấp phát bộ nhớ cho v13, rõ hơn là với hàm alloca. Vậy ta sẽ không cần quan tâm đến đoạn này nữa.

Đi đến bước encode dest:

```

for ( j = 0; ; j = encode(dest, j) )
{
    v10 = strlen(dest);
    if ( !(unsigned __int8)l(j, v10) )
        break;
}
v11 = strcmp(dest, "odt_sjtfnb_jc_c_fiajb_he_ciu_h_nkn_atvfjp");

```

- Trước tiên ta cần làm rõ hàm l ở đây:

```

1 __int64 __fastcall su(unsigned int a1, int a2)
2 {
3     if ( (unsigned __int8)eq(a2, 0) )
4         return a1;
5     if ( (unsigned __int8)eq(a1, 0) )
6         return 0LL;
7     return su(a1 - 1, (unsigned int)(a2 - 1));
8 }

```

Hàm su thực chất chỉ là su(a2,1) sẽ return a2-1

```

1 __int64 __fastcall l(unsigned int a1, unsigned int a2)
2 {
3     unsigned int v3; // ebx
4     unsigned int v4; // eax
5
6     if ( (unsigned __int8)eq(a1, 0LL) && (unsigned __int8)eq(a2, 0LL) )
7         return 0LL;
8     if ( (unsigned __int8)eq(a2, 0LL) )
9         return 0LL;
10    if ( (unsigned __int8)eq(a1, 0LL) )
11        return 1LL;
12    v3 = su(a2, 1LL);
13    v4 = su(a1, 1LL);
14    return l(v4, v3);
15 }

```

Hàm l sẽ thực hiện:

- Nếu a1 và a2 bằng 0 thì trả về false.
- Nếu a2 = 0 trả về false.
- Nếu a1 = 0 thì trả về 1.
- Sau đó nó giảm a1 và a2 1 đơn vị. Rồi gọi đệ quy với tham số đảo lại l(a2-1,a1-1) Chỉ với như vậy thì khá khó hiểu, tuy nhiên nhìn sơ sơ thì nó check cho đến khi một trong hai bằng 0, ta thử chạy nó xem:

```

1  def l(a1,a2):
2      if a1==0 and a2==0:
3          return 0
4      if a2==0:
5          return 0
6      if a1==0:
7          return 1
8      v3 = a2-1
9      v4 = a1-1
10     return l(v4,v3)
11
12  def main():
13      a1 = int(input("a1: "))
14      a2 = int(input("a2: "))
15      print(bool(l(a1,a2)))
16  if __name__ == "__main__":
17      main()
18
19  # __int64 __fastcall l(unsigned int a1, unsigned int a2)
20  # {
21  #     unsigned int v3; // ebx
22  #     unsigned int v4; // eax
23
24  #     if ( (unsigned __int8)eq(a1, 0LL) && (unsigned __int8)eq(a2, 0LL) )
25  #         return 0LL;
26  #     if ( (unsigned __int8)eq(a2, 0LL) )
27  #         return 0LL;
28  #     if ( (unsigned __int8)eq(a1, 0LL) )
29  #         return 1LL;
30  #     v3 = su(a2, 1LL);
31  #     v4 = su(a1, 1LL);
32  #     return l(v4, v3);
33  # }

```

```

rayinaw@rune:/mnt/hgfs/vmware$ python3 l.py
a1: 4
a2: 5
True
rayinaw@rune:/mnt/hgfs/vmware$ python3 l.py
a1: 5
a2: 5
False
rayinaw@rune:/mnt/hgfs/vmware$ python3 l.py
a1: 6
a2: 5
False

```

Vậy ta có thể biết được hàm l() sẽ check $a1 < a2$ thì return true, còn $a1 \geq a2$ thì return false.

- Vậy ở đây:


```

59     for ( j = 0; ; j = encode(dest, j) )
60     {
61         v10 = strlen(dest);
62         if ( !(unsigned __int8)l(j, v10) )
63             break;
64     }

```

Nó sẽ thực hiện encode theo từng ký tự của dest và break ra khi nó chạy hết chuỗi thôi.

- Ta cùng xem nó encode như nào:

```

1  int64 __fastcall encode(__int64 a1, int a2)
2  {
3      char j; // al
4      char k; // al
5      char l; // al
6      char m; // al
7      unsigned int v6; // eax
8      char v9; // [rsp+12h] [rbp-3Eh]
9      char v10; // [rsp+13h] [rbp-3Dh]
10     int i; // [rsp+14h] [rbp-3Ch]
11     unsigned int v12; // [rsp+18h] [rbp-38h]
12     unsigned int v13; // [rsp+1Ch] [rbp-34h]
13     unsigned int v14; // [rsp+20h] [rbp-30h]
14     unsigned int v15; // [rsp+24h] [rbp-2Ch]
15     unsigned int v16; // [rsp+28h] [rbp-28h]
16     unsigned int v17; // [rsp+2Ch] [rbp-24h]
17     unsigned int v18; // [rsp+30h] [rbp-20h]
18     unsigned int v19; // [rsp+34h] [rbp-1Ch]
19     unsigned int v20; // [rsp+38h] [rbp-18h]
20     unsigned int v21; // [rsp+40h] [rbp-10h]
21     int v22; // [rsp+44h] [rbp-Ch]
22     unsigned int v23; // [rsp+48h] [rbp-8h]
23     int v24; // [rsp+4Ch] [rbp-4h]
24
25     while ( (unsigned __int8)eq(*(char *)(a2 + a1), 95) )
26         a2 = s(a2);
27     for ( i = s(a2); (unsigned __int8)eq(*(char *)(i + a1), 95); i = s(i) )
28         ;
29     v9 = *(_BYTE *)(a2 + a1);
30     v14 = 0;
31     for ( j = ::l(0, 5u); j; j = ::l(v14, 5u) )
32     {
33         v15 = 0;
34         for ( k = ::l(0, 5u); k; k = ::l(v15, 5u) )
35         {
36             v23 = ::m(v14, 5LL);
37             v24 = a(v23, v15);
38             if ( (unsigned __int8)eq(aAbcdefghijklmn[v24], v9) )
39             {
40                 v12 = v14;
41                 v13 = v15;
42             }
43             v15 = s(v15);

```

00001559 encode:1 (1559)

```

43     v15 = s(v15);
44 }
45 v14 = s(v14);
46 }
47 v10 = *(_BYTE *)(i + a1);
48 v18 = 0;
49 for ( l = ::l(0, 5u); l; l = ::l(v18, 5u) )
50 {
51     v19 = 0;
52     for ( m = ::l(0, 5u); m; m = ::l(v19, 5u) )
53     {
54         v21 = ::m(v18, 5LL);
55         v22 = a(v21, v19);
56         if ( (unsigned __int8)eq(aAbcdefghijklmn[v22], v10) )
57         {
58             v16 = v18;
59             v17 = v19;
60         }
61         v19 = s(v19);
62     }
63     v18 = s(v18);
64 }
65 v6 = ::m(v12, 5LL);
66 *(_BYTE *)(a1 + a2) = aVastbcdefghijk[(int)a(v6, v17)];
67 v20 = ::m(v16, 5LL);
68 *(_BYTE *)(a1 + i) = aCornfieldsabgh[(int)a(v20, v13)];
69 return s(i);
70 }

```

Lưu ý là đối số truyền vào a1 là địa chỉ của dest, và a2 là j.

Đầu tiên ta để ý đến cái này:

```

25 while ( (unsigned __int8)eq(*(char *)(a2 + a1), 95) )
26     a2 = s(a2);

```

Số 95 ở đây là mã ascii của dấu gạch dưới '_'.

```

1 __int64 __fastcall s(int a1)
2 {
3     return (unsigned int)(a1 + 1);
4 }

```

Hàm s chỉ là gán i=i+1, chương trình hơi lạ ha 😊

Vậy có nghĩa là nó thực hiện check nếu ký tự thứ j của dest là '_' thì nó sẽ tăng a2 (là j) lên 1 đơn vị, tăng cho đến khi nó gặp ký tự khác '_' thì thoát.

```

27 for ( i = s(a2); (unsigned __int8)eq(*(char *)(i + a1), 95); i = s(i) )
28     ;
29 v9 = *(_BYTE *)(a2 + a1);
30 v14 = 0;

```

Vòng lặp này thực hiện tìm ký tự khác với '_' tiếp theo.

Vậy tổng từ dòng 25 đến 30 nó thực hiện check nếu ký tự thứ j là '_' thì sẽ tìm đến ký tự tiếp theo khác '_', rồi gán v9 cho ký tự đó.

Còn vòng lặp for chỉ để tìm vị trí ký tự tiếp theo khác '_', gán ký tự cho v10. Và vị trí này sẽ return về cho j.

Tiếp theo ta xem nó encode từng ký tự như thế nào:

- Xem trước các hàm nhỏ ở trong nó. Hàm a ở đây thực hiện cộng hai số a1 và a2.

```
1 __int64 __fastcall a(unsigned int a1, unsigned int a2)
2 {
3     unsigned int v3; // ebx
4     unsigned int v4; // eax
5
6     if ( (unsigned __int8)eq(a2, 0) )
7         return a1;
8     v3 = su(a2, 1);
9     v4 = s(a1);
10    return a(v4, v3);
11 }
```

```
1 def a(a1,a2):
2     if a2==0:
3         return a1
4     v3 = a2-1
5     v4 = a1+1
6     return a(v4,v3)
7
8 def main():
9     a1 = int(input("a1: "))
10    a2 = int(input("a2: "))
11    print(a(a1,a2))
12    if __name__ == "__main__":
13        main()
14
15 # __int64 __fastcall a(unsigned int a1, unsigned int a2)
16 # {
17 #     unsigned int v3; // ebx
18 #     unsigned int v4; // eax
19
20 #     if ( (unsigned __int8)eq(a2, 0) )
21 #         return a1;
22 #     v3 = su(a2, 1);
23 #     v4 = s(a1);
24 #     return a(v4, v3);
25 # }
26
```

```

rayinaw@rune:/mnt/hgfs/vmware$ python3 a.py
a1: 1
a2: 2
3
rayinaw@rune:/mnt/hgfs/vmware$ python3 a.py
a1: 2
a2: 1
3
rayinaw@rune:/mnt/hgfs/vmware$ python3 a.py
a1: 3
a2: 4
7
rayinaw@rune:/mnt/hgfs/vmware$ python3 a.py
a1: 0
a2: 0
0

```

- Hàm m ở đây thực hiện nhân hai số a1 với a2:

```

1 _int64 __fastcall m(unsigned int a1, int a2)
2 {
3     unsigned int v3; // [rsp+18h] [rbp-8h]
4
5     if ( (unsigned __int8)eq(a2, 0) )
6         return 0LL;
7     v3 = m(a1, (unsigned int)(a2 - 1));
8     return (unsigned int)a(a1, v3);
9 }

```

```

1 def a(a1,a2):
2     if a2==0:
3         return a1
4     v3 = a2-1
5     v4 = a1+1
6     return a(v4,v3)
7
8 def m(a1,a2):
9     if a2==0:
10        return 0
11    v3=m(a1,a2-1)
12    return a(a1,v3)
13 def main():
14     a1 = int(input("a1: "))
15     a2 = int(input("a2: "))
16     print(m(a1,a2))
17 if __name__ == "__main__":
18     main()
19

```



```

rayinaw@rune:/mnt/hgfs/vmware$ python3 m.py
a1: 1
a2: 2
2
rayinaw@rune:/mnt/hgfs/vmware$ python3 m.py
a1: 3
a2: 4
12
rayinaw@rune:/mnt/hgfs/vmware$ python3 m.py
a1: 0
a2: 0
0
rayinaw@rune:/mnt/hgfs/vmware$ python3 m.py
a1: 2
a2: 5
10

```

Xem tiếp vòng for:

```

for ( j = ::l(0, 5u); j; j = ::l(v14, 5u) )
{
    v15 = 0;
    for ( k = ::l(0, 5u); k; k = ::l(v15, 5u) )
    {
        v23 = ::m(v14, 5LL);
        v24 = a(v23, v15);
        if ( (unsigned __int8)eq(aAbcdefghijklmn[v24], v9) )
        {
            v12 = v14;
            v13 = v15;
        }
        v15 = s(v15);
    }
    v14 = s(v14);
}

```

Vòng for này sẽ kiểu như này:

```

for v14 in range(5):
    for v15 in range(5):
        v24=v14*5+v15
        if(str[v24]==v9):
            v12 = v14
            v13 = v15

```

Thực chất nó chỉ quét qua tất cả các ký tự từ a-z:

```

.rodata:0000000000002000 db 0
.rodata:0000000000002008 aAbcdefghijklmn db 'abcdefghijklmnoprstuvwxyz',0
.rodata:0000000000002008 ; DATA XREF: encode+D9↑o

```

Test thử cho nó dễ hiểu nè:

```

3  str="abcdefghijklmnopqrstuvwxyz"
4  for v14 in range (5):
5      for v15 in range(5):
6          v24=v14*5+v15
7          print(str[v24], end=' ')

rayinaw@rune:/mnt/hgfs/vmware$ python3 for1.py
abcdefghijklmnopqrstuvwxyzrayinaw@rune:/mnt/hgfs/vmware$

```

Vòng for tiếp theo cũng tương tự:

```

47  v10 = *(_BYTE *) (i + a1);
48  v18 = 0;
49  for ( l = ::l(0, 5u); l; l = ::l(v18, 5u) )
50  {
51      v19 = 0;
52      for ( m = ::l(0, 5u); m; m = ::l(v19, 5u) )
53      {
54          v21 = ::m(v18, 5LL);
55          v22 = a(v21, v19);
56          if ( (unsigned __int8)eq(aAbcdefghijklmn[v22], v10) )
57          {
58              v16 = v18;
59              v17 = v19;
60          }
61          v19 = s(v19);
62      }
63      v18 = s(v18);
64  }

```

Tóm lại hai vòng for này sẽ thực hiện:

- Với vòng for đầu tiên, tìm vị trí của ký tự v9 trong bảng chữ cái (v24).
Tương ứng với biểu thức $v24 = v14 * 5 + v15$
Gán $v12 = v14$, $v13 = v15$
- Với vòng for thứ hai, nó cũng tương tự tìm vị trí của ký tự v10 trong bảng chữ cái (v22).
 $V22 = v18 * 5 + v19$
Gán $v16 = v18$, $v17 = v19$

Bây giờ đến công đoạn cuối cùng, encode dest, hàm m là nhân, hàm a là tổng:

```

65  v6 = ::m(v12, 5LL);
66  *(_BYTE *) (a1 + a2) = aVastbcdefghijk[(int)a(v6, v17)];
67  v20 = ::m(v16, 5LL);
68  *(_BYTE *) (a1 + i) = aCornfieldsabgh[(int)a(v20, v13)];
69  return s(i);

```

- Ký tự thứ $dest[a2] = aVastbcdefghijk[v12 * 5 + v17]$
- Ký tự thứ $dest[i] = aCornfieldsabgh[v16 * 5 + v13]$

Để thấy nó encode word by word. Vậy nên ta có thể lấy ra được từng cặp (v12,v17) và (v16,v13).

Reverse chạy chỗ này:

```
dest_reverse='odt_sjtfnb_jc_c_fiajb_he_ciuh_nkn_atvfjp'
aVastbcdefghijk='vastbcdefghijklmnopruwxyz'
aCornfieldsabgh='cornfieldsabghjkmptuvwxyz'
str="abcdefghijklmnoprstuvwxyz"
dest = ['']*50

def find_v12_v17(chr):
    for x in range (5):
        for y in range (5):
            if(chr==aVastbcdefghijk[x*5+y]):
                return (x,y)
def find_v16_v13(chr):
    for x in range (5):
        for y in range (5):
            if(chr==aCornfieldsabgh[x*5+y]):
                return (x,y)

def decode(i):
    while i < len(dest_reverse) and dest_reverse[i] == '_':
        dest[i]='_'
        i += 1
    j = i + 1
    while j < len(dest_reverse) and dest_reverse[j] == '_':
        dest[j]='_'
        j += 1
    if(j==len(dest_reverse)):
        return j
    v9=dest_reverse[i]
    v10=dest_reverse[j]
    v12,v17 = find_v12_v17(v9)
    v16,v13 = find_v16_v13(v10)
    dest[i] = str[5*v12+v13]
    dest[j] = str[5*v16+v17]
    return j+1

def main():
    i=0
```

```
while i < len(dest_reverse):  
    i=decode(i)  
    print(''.join(dest))  
  
if __name__ == "__main__":  
    main()
```

Kết quả:

```
rayinaw@rune:~/vmware$ python3 dump2.py  
'the_inside_of_a_field_of_corn_and_dreams'  
rayinaw@rune:~/vmware$ ./hard_chal  
[$] Enter your input in the form: words_with_underscores_and_letters: the_inside_of_a_field_of_corn_and_dreams  
[$] Correct!  
[$] uiuctf{the_inside_of_a_field_of_corn_and_dreams}
```