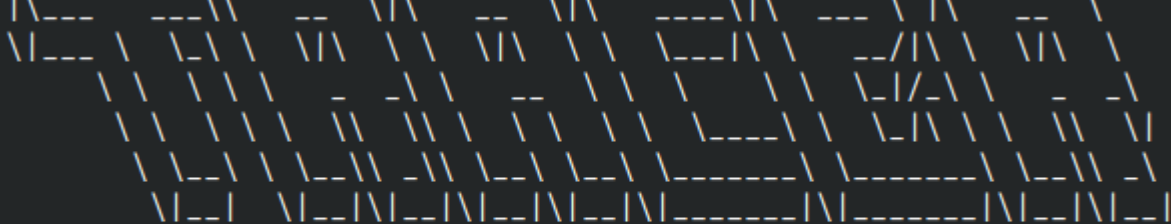


trace

首先試著執行題目的檔案

```
$ ./trace_63ae7693f94d1f0b
```



```
Please  
Give me flag  
flag{hello_world}  
Try harder :(  
$ █
```

似乎沒有什麼有用的資訊

Decompile

使用ghidra反組譯

首先先找到entry，然後跟着entry到main

```

1 void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)
2 {
3     undefined8 in_stack_00000000;
4     undefined auStack8 [8];
5     __libc_start_main(main,in_stack_00000000,&stack0x00000008,0,0,param_3,auStack8);
6     do {
7         // WARNING: Do nothing block with infinite loop
8     } while( true );
9 }
10
11
12
13

```

```
main:
```

cs_2022_fall_ouo

entry:

```
1
2 void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)
3
4 {
5     undefined8 in_stack_00000000;
6     undefined auStack8 [8];
7
8     __libc_start_main(FUN_004012be,in_stack_00000000,&stack0x00000008,0,0,param_3,auStack8);
9     do {
10         // WARNING: Do nothing block with infinite loop
11     } while( true );
12 }
13
```

跟著往下走

```
1
2 void FUN_004012be(void)
3
4 {
5     code *pcVar1;
6
7     func_0xffffffffdeedd1ba();
8     pcVar1 = (code *)swi(3);
9     (*pcVar1)();
10    return;
11 }
12
```

結果無法繼續追蹤，因為不存在func_0xffffffffdeedd1ba這個位置

Discovery

瀏覽整個反組譯過的程式，在0x004011d8看到了一些資料

```

                                undefined FUN_004011ca()
                                AL:1          <RETURN>
                                FUN_004011ca
                                XREF[2]:      00e
004011ca f3 0f 1e  ENDBR64
           fa
004011ce 55        PUSH    RBP
004011cf 48 89 e5   MOV     RBP,RSP
004011d2 e8 ef be   CALL    SUB_ffffffffdeedd0c6
           ad de
004011d7 cc        INT3
004011d8 cb        ??      CBh
004011d9 e8        ??      E8h
004011da c7        ??      C7h
004011db 45        ??      45h    E
004011dc fc        ??      FCh
004011dd 00        ??      00h
004011de 00        ??      00h
004011df 00        ??      00h
004011e0 00        ??      00h
004011e1 eb        ??      EBh
004011e2 28        ??      28h    (
004011e3 8b        ??      8Bh
004011e4 45        ??      45h    E
004011e5 fc        ??      FCh
004011e6 48        ??      48h    H
004011e7 98        ??      98h
004011e8 48        ??      48h    H
004011e9 8d        ??      8Dh
004011ea 15        ??      15h
004011eb 61        ??      61h    a
004011ec 2e        ??      2Eh    .
004011ed 00        ??      00h
004011ee 00        ??      00h
004011ef 0f        ??      0Fh
004011f0 b6        ??      B6h
004011f1 04        ??      04h
004011f2 10        ??      10h
004011f3 83        ??      83h
004011f4 f0        ??      F0h
004011f5 71        ??      71h    a
```

但看起來怪怪的，因此試着disassemble看看

```
004011ca f3 0f 1e FUN_004011ca XREF[2]: 00402070, 00402150
    ENDBR64
004011ce 55      PUSH    RBP
004011cf 48 89 e5 MOV     RBP, RSP
004011d2 e8 ef be CALL    SUB_ffffffdeedd0c6
    ad de
004011d7 cc      INT3
004011d8 cb      RETF
004011d9 e8 c7 45 CALL    SUB_013c57a5
    fc 00
004011de 00 00    ADD     byte ptr [RAX], AL
004011e0 00 eb    ADD     BL, CH
    LAB_004011e2+1
004011e2 28 8b 45 SUB     byte ptr [RBX + -0x67b703bb], CL XREF[0,1]: 00401211(j)
    fc 48 98
004011e8 48 8d 15 LEA     RDX, [DAT_00404050] = 37h 7
    61 2e 00
004011ef 0f b6 04 MOVZX   EAX, byte ptr [RAX + RDX*0x1]=>DAT_00404050 = 37h 7
    10
004011f3 83 f0 71 XOR     EAX, 0x71
004011f6 89 c1    MOV     ECX, EAX
004011f8 8b 45 fc MOV     EAX, dword ptr [RBP + -0x4]
004011fb 48 98    CDQE
004011fd 48 8d 15 LEA     RDX, [DAT_00404050] = 37h 7
    4c 2e 00
00401204 88 0c 10 MOV     byte ptr [RAX + RDX*0x1]=>DAT_00404050, CL = 37h 7
00401207 83 45 fc ADD     dword ptr [RBP + -0x4], 0x1
    01
0040120b 8b 45 fc MOV     EAX, dword ptr [RBP + -0x4]
0040120e 83 f8 16 CMP     EAX, 0x16
00401211 76 d0    JBE     LAB_004011e2+1
00401213 e8 ef be CALL    SUB_ffffffdeedd107
    ad de
00401218 cc      INT3
00401219 cb      RETF
0040121a e8      ??     E8h
0040121b 90      NOP
0040121c 5d      POP     RBP
0040121d c3      RET

//
4
5 void UndefinedFunction_004011d9(void)
6
7 {
8     char *pcVar1;
9     code *pcVar2;
10    char *pcVar3;
11    undefined2 uVar4;
12    uint in_ECX;
13    char unaff_BL;
14    undefined7 unaff_00000019;
15    long unaff_RBP;
16
17    do {
18        uVar4 = (undefined2)(in_ECX >> 8);
19        pcVar3 = (char *)func_0x013c57a5();
20        *pcVar3 = *pcVar3 + (char)pcVar3;
21        unaff_BL = unaff_BL + (char)((ushort)uVar4 >> 8);
22        pcVar1 = (char *)CONCAT71(unaff_00000019, unaff_BL) + -0x67b703bb);
23        *pcVar1 = *pcVar1 - (char)uVar4;
24        in_ECX = (byte)pcVar3[0x404050] ^ 0x71;
25        (&DAT_00404050)[*(int *) (unaff_RBP + -4)] = (char)in_ECX;
26        *(int *) (unaff_RBP + -4) = *(int *) (unaff_RBP + -4) + 1;
27    } while (*(uint *) (unaff_RBP + -4) < 0x17);
28    func_0xffffffffdeedd107();
29    pcVar2 = (code *)swi(3);
30    (*pcVar2)();
31    return;
32 }
33
```

結果成功解出東西。
但沒有看的很懂，因此繼續往後找

在0x00401242又看到了一些不明資料，嘗試disassemble

```

ad de
00401241 cc      INT3
00401242 cb      ??      CBh
00401243 e8      ??      E8h
00401244 48      ??      48h      H
00401245 8d      ??      8Dh
00401246 85      ??      85h
00401247 f0      ??      F0h
00401248 fe      ??      FEh
00401249 ff      ??      FFh
0040124a ff      ??      FFh
0040124b 48      ??      48h      H
0040124c 89      ??      89h
0040124d c6      ??      C6h
0040124e 48      ??      48h      H
0040124f 8d      ??      8Dh
00401250 05      ??      05h
00401251 c3      ??      C3h
00401252 0d      ??      0Dh
00401253 00      ??      00h
00401254 00      ??      00h
00401255 48      ??      48h      H
00401256 89      ??      89h
00401257 c7      ??      C7h
00401258 b8      ??      B8h
00401259 00      ??      00h
0040125a 00      ??      00h
0040125b 00      ??      00h
0040125c 00      ??      00h
0040125d e8      ??      E8h
0040125e 3e      ??      3Eh      >
0040125f fe      ??      FEh
00401260 ff      ??      FFh
00401261 ff      ??      FFh
00401262 e8      ??      E8h
00401263 ef      ??      EFh
00401264 be      ??      BEh
00401265 ad      ??      ADh
00401266 de      ??      DEh

```

結果也成功解出東西，下面是解碼出來的組合語言。

```
00401262 e8 ef be CALL SUB_ffffffffffdeedd1b6
ad de
00401267 cc INT3
00401268 cb RETF
00401269 e8 48 8d CALL SUB_ffffffffff0c59fb6
85 f0
0040126e fe ?? FEh
0040126f ff ?? FFh
00401270 ff 48 8d DEC dword ptr [RAX + -0x73]
00401273 15 d8 2d ADC EAX,0x2dd8
00 00
00401278 48 89 d6 MOV RSI,RDX
0040127b 48 89 c7 MOV RDI,RAX
0040127e e8 0d fe CALL FUN_00401090
ff ff undefined FUN_00401090
00401283 85 c0 TEST EAX,EAX
00401285 75 11 JNZ LAB_00401298
00401287 48 8d 05 LEA RAX,[s_Well_done!_0040201b] = "Well done!"
00 00
0040128e 48 89 c7 MOV RDI=>s_Well_done!_0040201b,RAX = "Well done!"
00401291 e8 da fd CALL <EXTERNAL>::puts int puts(char *)
ff ff
00401296 eb 0f JMP LAB_004012a7
LAB_00401298 XREF[1]: 00401285(j)
00401298 48 8d 05 LEA RAX,[s_Try_harder!:_00402026] = "Try harder !"
87 0d 00
0040129f 48 89 c7 MOV RDI=>s_Try_harder!:_00402026,RAX = "Try harder !"
004012a2 e8 c9 fd CALL <EXTERNAL>::puts int puts(char *)
ff ff
LAB_004012a7 XREF[1]: 00401296(j)
004012a7 90 NOP
004012a8 48 8b 45 MOV RAX,qword ptr [RBP + -0x8]
f8
004012ac 64 48 2b SUB RAX,qword ptr FS:[0x28]
04 25 28
00 00 00
004012b5 74 05 JZ LAB_004012bc
004012b7 e8 c4 fd CALL FUN_00401080
ff ff undefined FUN_00401080
```

```
void UndefinedFunction_00401270(undefined8 param_1,undefined8 param_2,undefined8 param_3)
{
    int iVar1;
    long in_RAX;
    long unaff_RBP;
    long in_FS_OFFSET;
    byte in_CF;

    *(int *)(in_RAX + -0x73) = *(int *)(in_RAX + -0x73) + -1;
    iVar1 = FUN_00401090((int)in_RAX + 0x2dd8 + (uint)in_CF,param_3);
    if (iVar1 == 0) {
        puts("Well done!");
    }
    else {
        puts("Try harder :(");
    }
    if ((*long *)(&unaff_RBP + -8) != (*long *)(&in_FS_OFFSET + 0x28)) {
        FUN_00401080();
    }
    return;
}
```

雖然出現了一些題目關鍵的字串，但沒有看的很懂，因此繼續往後找

Suspicious data

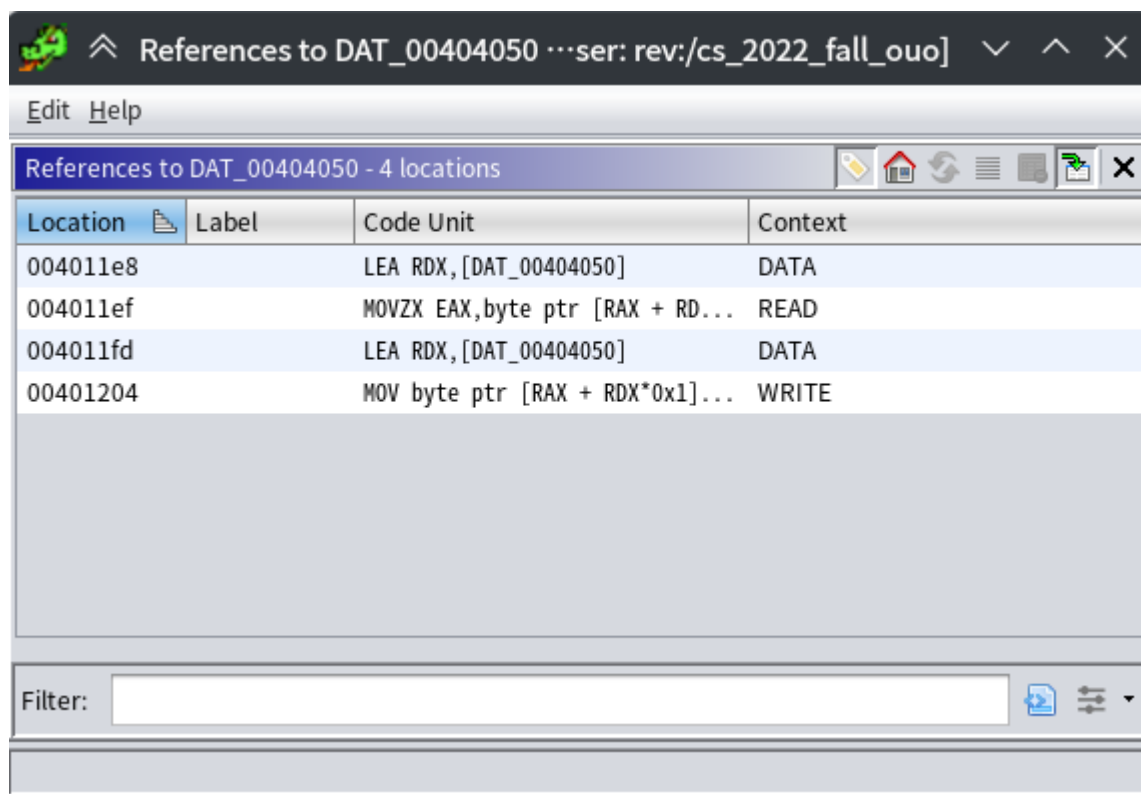
在0x00404050位置發現可疑的資料

因為出現很多printable的ascii以及最後一個byte是0x00，長度也算合理，因此猜測是flag

DAT_00404050

00404050	37	??	37h	7
00404051	3d	??	3Dh	=
00404052	30	??	30h	0
00404053	36	??	36h	6
00404054	0a	??	0Ah	
00404055	25	??	25h	%
00404056	03	??	03h	
00404057	30	??	30h	0
00404058	12	??	12h	
00404059	42	??	42h	B
0040405a	2e	??	2Eh	.
0040405b	3c	??	3Ch	<
0040405c	42	??	42h	B
0040405d	2e	??	2Eh	.
0040405e	40	??	40h	@
0040405f	37	??	37h	7
00404060	2e	??	2Eh	.
00404061	24	??	24h	\$
00404062	2e	??	2Eh	.
00404063	12	??	12h	
00404064	30	??	30h	0
00404065	3f	??	3Fh	?
00404066	0c	??	0Ch	
00404067	00	??	00h	

使用ghidra的功能尋找這個位置的reference



結果在剛才disassemble的地方找到reference

004011e8	fc 48 98 48 8d 15 61 2e 00 00	LEA	RDX,[DAT_00404050]	= 37h	7
004011ef	0f b6 04 10	MOVZX	EAX,byte ptr [RAX + RDX*0x1]=>DAT_00404050	= 37h	7
004011f3	83 f0 71	XOR	EAX,0x71		
004011f6	89 c1	MOV	ECX,EAX		
004011f8	8b 45 fc	MOV	EAX,dword ptr [RBP + -0x4]		
004011fb	48 98	CDQE			
004011fd	48 8d 15 ...	LEA	RDX,[DAT_00404050]	= 37h	7

這邊反編譯出的程式碼如下:

```
// WARNING: Instruction at (ram,0x004011e3) overlaps instruction at
// (ram,0x004011e2)
//
void UndefinedFunction_004011d9(void) {
    char *pcVar1;
    code *pcVar2;
```

```

char *pcVar3;
undefined2 uVar4;
uint in_ECX;
char unaff_BL;
undefined7 unaff_00000019;
long unaff_RBP;

do {
    uVar4 = (undefined2)(in_ECX >> 8);
    pcVar3 = (char *)func_0x013c57a5();
    *pcVar3 = *pcVar3 + (char)pcVar3;
    unaff_BL = unaff_BL + (char)((ushort)uVar4 >> 8);
    pcVar1 = (char *) (CONCAT71(unaff_00000019, unaff_BL) + -0x67b703bb);
    *pcVar1 = *pcVar1 - (char)uVar4;
    in_ECX = (byte)pcVar3[0x404050] ^ 0x71;
    (&DAT_00404050)[*(int *) (unaff_RBP + -4)] = (char)in_ECX;
    *(int *) (unaff_RBP + -4) = *(int *) (unaff_RBP + -4) + 1;
} while (*(uint *) (unaff_RBP + -4) < 0x17);
func_0xffffffffdeedd107();
pcVar2 = (code *)swi(3);
(*pcVar2)();
return;
}

```

雖然整段程式碼非常雜亂很難看懂，仍舊看到了一行關鍵程式碼：

```
in_ECX = (byte)pcVar3[0x404050] ^ 0x71;
```

抱著疑心，把0x404050的位置的資料全部拿去XOR 0x71試看看

Solve

使用python協助做XOR

```

data = [0x37, 0x3d, 0x30, 0x36, 0x0a, 0x25, 0x03, 0x30, 0x12, 0x42, 0x2e,
        ↪ 0x3c,
        0x42, 0x2e, 0x40, 0x37, 0x2e, 0x24, 0x2e, 0x12, 0x30, 0x3f, 0x0c,
        ↪ 0x00]
res = [chr(i ^ 0x71) for i in data]
print("".join(res))

```

執行結果：

```
$ python3 sol.py  
FLAG{TrAc3_M3_1F_U_cAN}q
```

找到flag了。