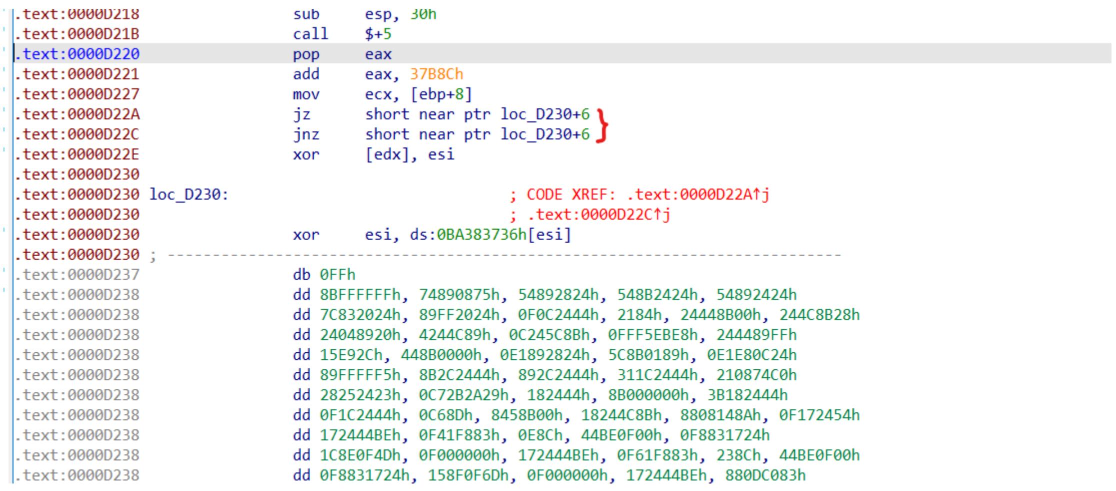
Lab10 Advanced Reverse

STU ID: 20307130044
Your Flag: FLAG{gorgeoushomework}
Your Bonus Flag：FLAG{thisissecretflag}

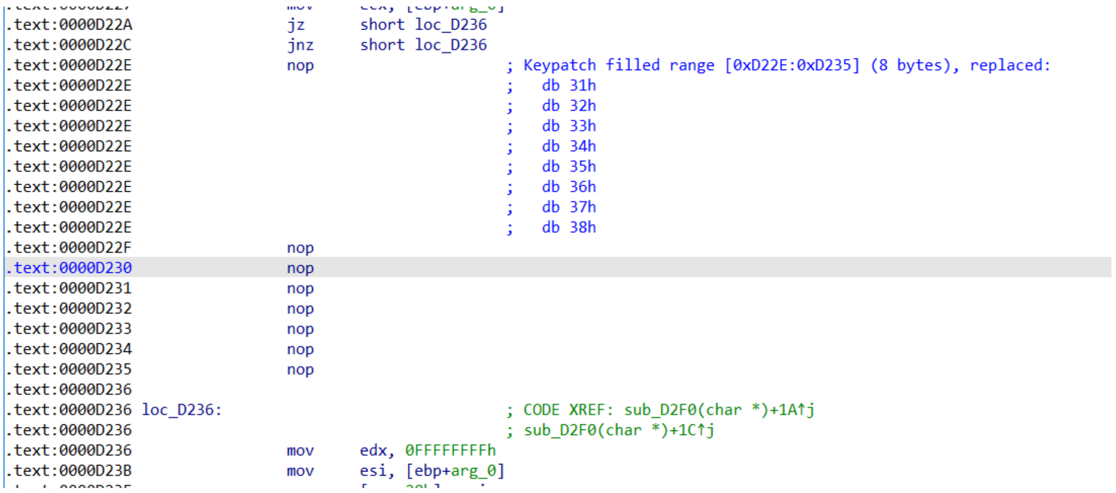
Analysis Process Breakdown:

改的地方挺多的，所以主要把每一类的反反汇编介绍一下：
第一种



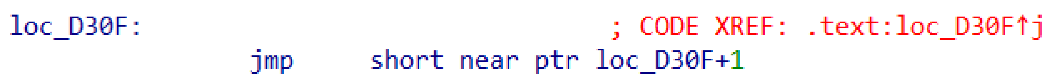
```
.text:0000D218                 sub     esp, 30h
.text:0000D21B                 call    $+5
.text:0000D220                 pop     eax
.text:0000D221                 add     eax, 37B8Ch
.text:0000D227                 mov     ecx, [ebp+8]
.text:0000D22A                 jz      short near ptr loc_D230+6
.text:0000D22C                 jnz     short near ptr loc_D230+6
.text:0000D22E                 xor     [edx], esi
.text:0000D230
.text:0000D230 loc_D230:                               ; CODE XREF: .text:0000D22A↑j
.text:0000D230                                         ; .text:0000D22C↑j
.text:0000D230                 xor     esi, ds:0BA383736h[esi]
.text:0000D230 ; ---------------------------------------------------------------------------
.text:0000D237                 db 0FFh
.text:0000D238                 dd 8BFFFFFFh, 74890875h, 54892824h, 548B2424h, 54892424h
.text:0000D238                 dd 7C832024h, 89FF2024h, 0F0C2444h, 2184h, 24448B00h, 244C8B28h
.text:0000D238                 dd 24048920h, 4244C89h, 0C245C8Bh, 0FFF5EBE8h, 244489FFh
.text:0000D238                 dd 15E92Ch, 448B0000h, 0E1892824h, 5C8B0189h, 0E1E80C24h
.text:0000D238                 dd 89FFFFF5h, 8B2C2444h, 892C2444h, 311C2444h, 210874C0h
.text:0000D238                 dd 28252423h, 0C72B2A29h, 182444h, 8B000000h, 3B182444h
.text:0000D238                 dd 0F1C2444h, 0C68Dh, 8458B00h, 18244C8Bh, 8808148Ah, 0F172454h
.text:0000D238                 dd 172444BEh, 0F41F883h, 0E8Ch, 44BE0F00h, 0F8831724h
.text:0000D238                 dd 1C8E0F4Dh, 0F000000h, 172444BEh, 0F61F883h, 238Ch, 44BE0F00h
.text:0000D238                 dd 0F8831724h, 158F0F6Dh, 0F000000h, 172444BEh, 880DC083h
```

这个是 jz 和 jnz 必定跳转，所以下面的指令是干扰的，直接改成数据，然后改 nop 就好了



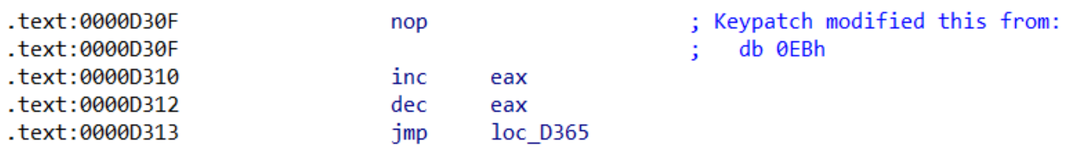
```
.text:0000D22A                 jz      short loc_D236
.text:0000D22C                 jnz     short loc_D236
.text:0000D22E                 nop                     ; Keypatch filled range [0xD22E:0xD235] (8 bytes), replaced:
.text:0000D22E                                         ;   db 31h
.text:0000D22E                                         ;   db 32h
.text:0000D22E                                         ;   db 33h
.text:0000D22E                                         ;   db 34h
.text:0000D22E                                         ;   db 35h
.text:0000D22E                                         ;   db 36h
.text:0000D22E                                         ;   db 37h
.text:0000D22E                                         ;   db 38h
.text:0000D22F                 nop
.text:0000D230                 nop
.text:0000D231                 nop
.text:0000D232                 nop
.text:0000D233                 nop
.text:0000D234                 nop
.text:0000D235                 nop
.text:0000D236
.text:0000D236 loc_D236:                               ; CODE XREF: sub_D2F0(char *)+1A↑j
.text:0000D236                                         ; sub_D2F0(char *)+1C↑j
.text:0000D236                 mov     edx, 0FFFFFFFFh
.text:0000D23B                 mov     esi, [ebp+arg_0]
```

第二种：



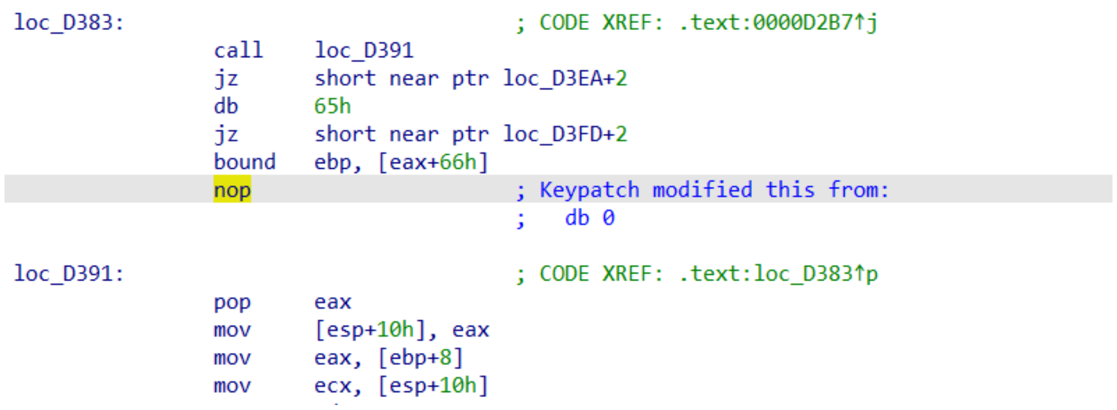
```
loc_D30F:                               ; CODE XREF: .text:loc_D30F↑j
                jmp     short near ptr loc_D30F+1
```

这种自己跳自己后面的指令，和直接执行没有区别，需要改成 data 后然后把后面的变为正常指令



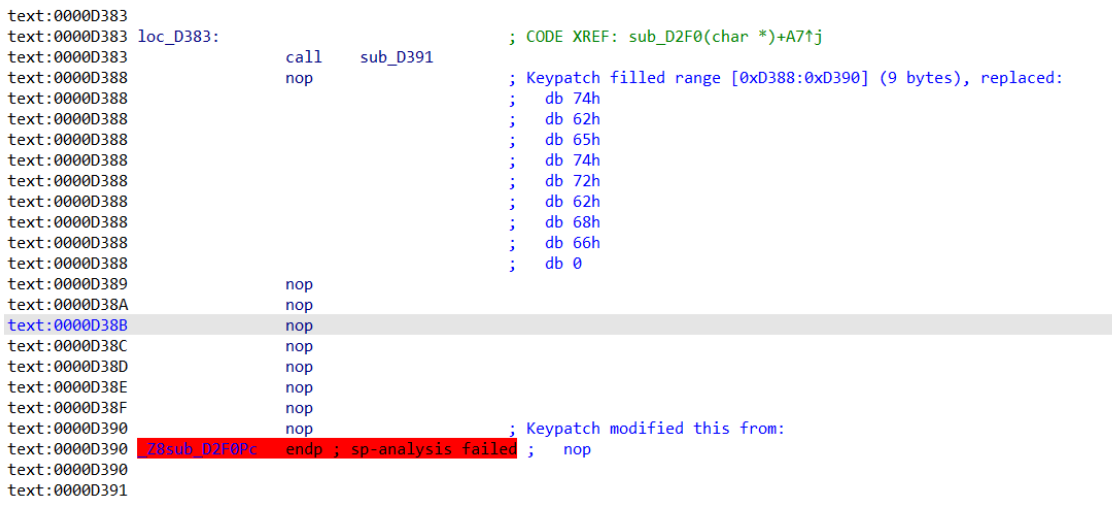
```
.text:0000D30F                 nop                     ; Keypatch modified this from:
.text:0000D30F                                         ;   db 0EBh
.text:0000D310                 inc     eax
.text:0000D312                 dec     eax
.text:0000D313                 jmp     loc_D365
```

第三种：



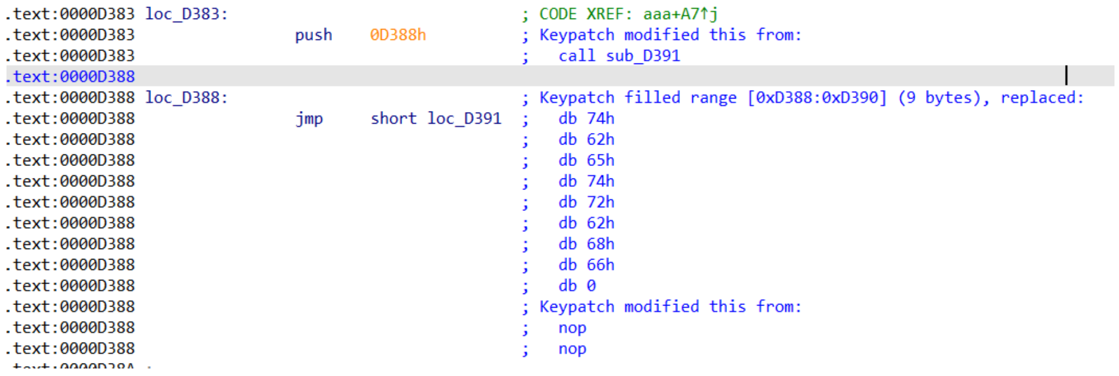
```
 loc_D383:                              ; CODE XREF: .text:0000D2B7↑j
                 call    loc_D391
                 jz      short near ptr loc_D3EA+2
                 db      65h
                 jz      short near ptr loc_D3FD+2
                 bound   ebp, [eax+66h]
                 nop                    ; Keypatch modified this from:
                                        ;   db 0

 loc_D391:                              ; CODE XREF: .text:loc_D383↑p
                 pop     eax
                 mov     [esp+10h], eax
                 mov     eax, [ebp+8]
                 mov     ecx, [esp+10h]
```

call loc_D391 然后是 pop eax 这一段指令就相当于跳过这一段了，所以 call 到 pop 的都是没用的，但是我注释掉后，反编译出现了如下问题



```
text:0000D383
text:0000D383 loc_D383:                             ; CODE XREF: sub_D2F0(char *)+A7↑j
text:0000D383                 call    sub_D391
text:0000D388                 nop                   ; Keypatch filled range [0xD388:0xD390] (9 bytes), replaced:
text:0000D388                                       ;   db 74h
text:0000D388                                       ;   db 62h
text:0000D388                                       ;   db 65h
text:0000D388                                       ;   db 74h
text:0000D388                                       ;   db 72h
text:0000D388                                       ;   db 62h
text:0000D388                                       ;   db 68h
text:0000D388                                       ;   db 66h
text:0000D388                                       ;   db 0
text:0000D389                 nop
text:0000D38A                 nop
text:0000D38B                 nop
text:0000D38C                 nop
text:0000D38D                 nop
text:0000D38E                 nop
text:0000D38F                 nop
text:0000D390                 nop                   ; Keypatch modified this from:
text:0000D390 _Z8sub_D2F0Pc   endp ; sp-analysis failed ;   nop
text:0000D390
text:0000D391
```

因为函数缺少一个 retn，所以堆栈似乎有问题，因为有 call，目标会认为函数，而只有一个 retn（也不一定，反正有点没搞清）， 我就尝试直接把 call 指令改为 push + jmp 的组合，就不报错了



```
.text:0000D383 loc_D383:                            ; CODE XREF: aaa+A7↑j
.text:0000D383                 push    0D388h       ; Keypatch modified this from:
.text:0000D383                                      ;   call sub_D391
.text:0000D388
.text:0000D388 loc_D388:                            ; Keypatch filled range [0xD388:0xD390] (9 bytes), replaced:
.text:0000D388                 jmp     short loc_D391 ;   db 74h
.text:0000D388                                      ;   db 62h
.text:0000D388                                      ;   db 65h
.text:0000D388                                      ;   db 74h
.text:0000D388                                      ;   db 72h
.text:0000D388                                      ;   db 62h
.text:0000D388                                      ;   db 68h
.text:0000D388                                      ;   db 66h
.text:0000D388                                      ;   db 0
.text:0000D388                                      ; Keypatch modified this from:
.text:0000D388                                      ;   nop
.text:0000D388                                      ;   nop
```

下面这是第四种，因为我提前 keypatch 掉了 但是不方便改回来了，这是 call + 无用代码 + retn 的混淆指令，让反编译器以为函数结束 retn 了，其实什么事情都没干，nop 掉就好了

```
.text:0000D504                 nop                     ; Keypatch filled range [0xD504:0xD50F] (12 bytes), replaced:
.text:0000D504                                         ;   call $+5
.text:0000D504                                         ;   add dword ptr [esp], 7
.text:0000D504                                         ;   xor eax, eax
.text:0000D504                                         ;   retn
.text:0000D505                 nop
.text:0000D506                 nop
.text:0000D507                 nop
.text:0000D508                 nop
.text:0000D509                 nop
.text:0000D50A                 nop
.text:0000D50B                 nop
.text:0000D50C                 nop
.text:0000D50D                 nop
.text:0000D50E                 nop
.text:0000D50F                 nop
.text:0000D510                 mov     dword ptr [esp+30h], 0
.text:0000D518
.text:0000D518 loc_D518:                               ; CODE XREF: sub_D4B0(char *)+1DE↓j
```

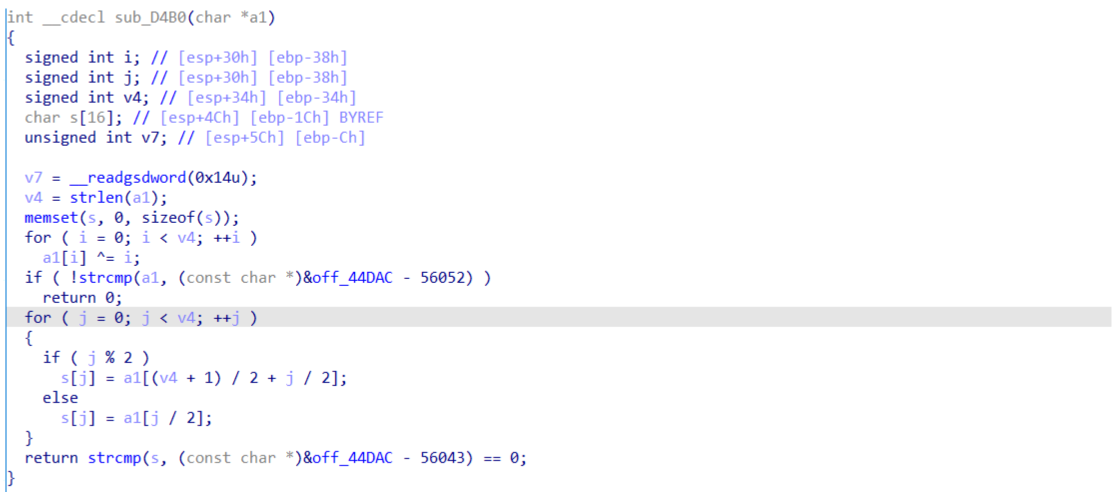
下面是反编译出来的两个正确的函数

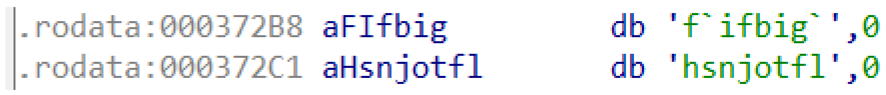
第一个函数的地址是 8 长的字符串，tbetrbhf

```
BOOL4 __cdecl aaa(const char *a1)
{
  char v2; // [esp+17h] [ebp-21h]
  signed int i; // [esp+18h] [ebp-20h]
  signed int v4; // [esp+1Ch] [ebp-1Ch]

  v4 = strlen(a1);
  for ( i = 0; i < v4; ++i )
  {
    v2 = a1[i];
    if ( (v2 < 65 || v2 > 77) && (v2 < 97 || v2 > 109) )
    {
      if ( v2 >= 78 && v2 <= 90 || v2 >= 110 && v2 <= 122 )
        v2 -= 13;
    }
    else
    {
      v2 += 13;
    }
    a1[i] = v2;
  }
  return strcmp(a1, (const char *)&loc_D388) == 0;
}
```

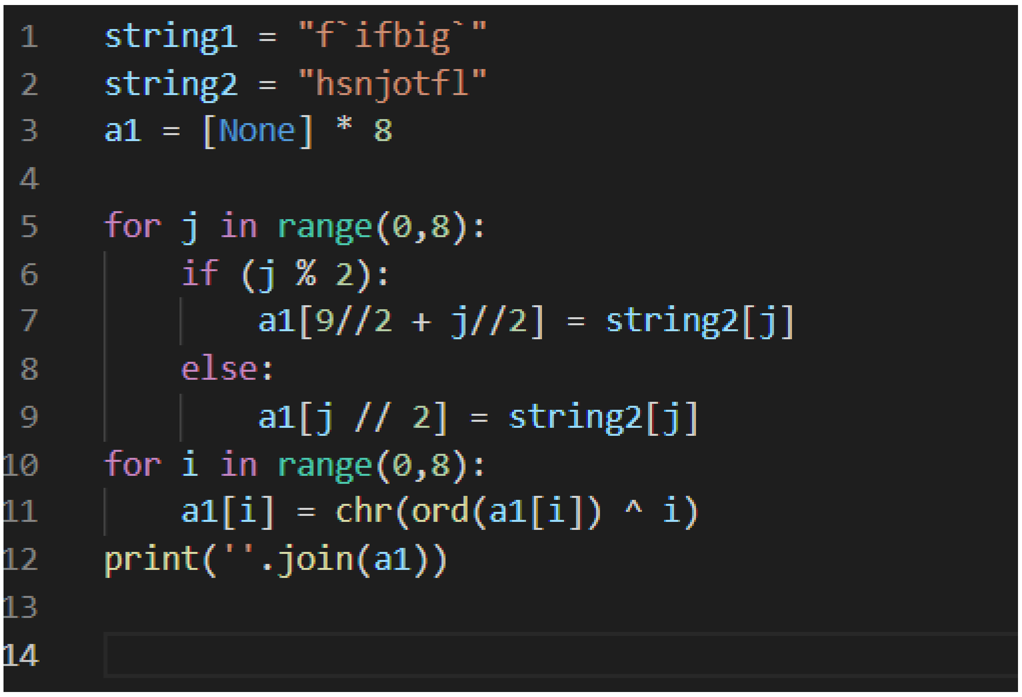
```
;    db 74h
;    db 62h
;    db 65h
;    db 74h
;    db 72h
;    db 62h
;    db 68h
;    db 66h
```

第二个函数有两个字符串的 strcmp，但是感觉第一个比较很怪，相等才返回 0，很奇怪，似乎有 bonus 的味道(其实是假的)


```c
int __cdecl sub_D4B0(char *a1)
{
  signed int i; // [esp+30h] [ebp-38h]
  signed int j; // [esp+30h] [ebp-38h]
  signed int v4; // [esp+34h] [ebp-34h]
  char s[16]; // [esp+4Ch] [ebp-1Ch] BYREF
  unsigned int v7; // [esp+5Ch] [ebp-Ch]

  v7 = __readgsdword(0x14u);
  v4 = strlen(a1);
  memset(s, 0, sizeof(s));
  for ( i = 0; i < v4; ++i )
    a1[i] ^= i;
  if ( !strcmp(a1, (const char *)&off_44DAC - 56052) )
    return 0;
  for ( j = 0; j < v4; ++j )
  {
    if ( j % 2 )
      s[j] = a1[(v4 + 1) / 2 + j / 2];
    else
      s[j] = a1[j / 2];
  }
  return strcmp(s, (const char *)&off_44DAC - 56043) == 0;
}
```

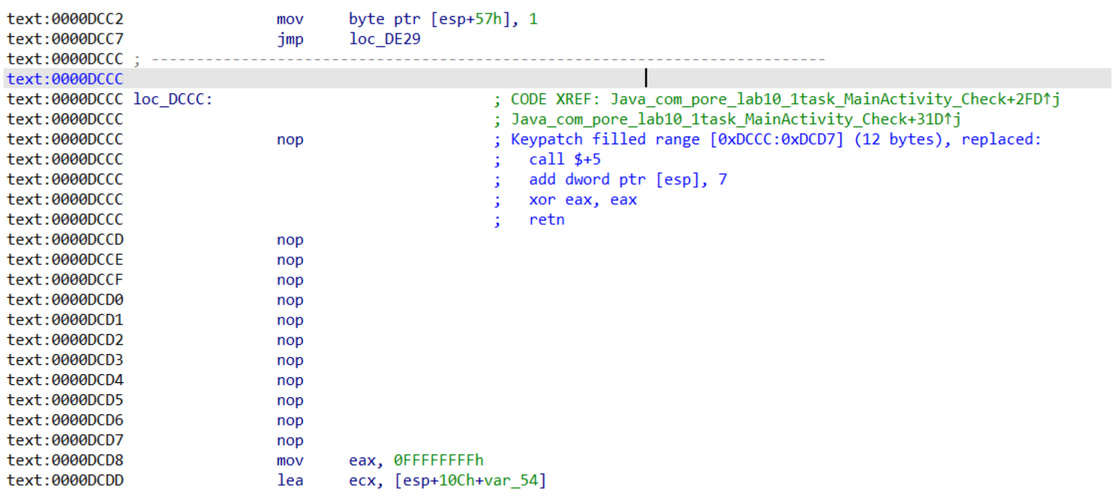

```
.rodata:000372B8 aFIfbig         db 'f`ifbig`',0
.rodata:000372C1 aHsnjotfl       db 'hsnjotfl',0
```

不论怎么样，先把基本的解出来，都比较简单的解密，就不详细给过程了吧……


```python
string1 = "f`ifbig`"
string2 = "hsnjotfl"
a1 = [None] * 8

for j in range(0,8):
    if (j % 2):
        a1[9//2 + j//2] = string2[j]
    else:
        a1[j // 2] = string2[j]
for i in range(0,8):
    a1[i] = chr(ord(a1[i]) ^ i)
print(''.join(a1))
```

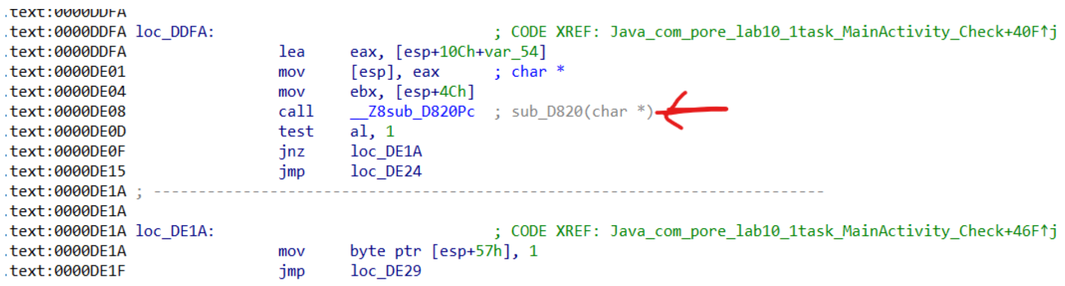
答案是 gorgeoushomework！ 华丽的作业(•̀ ω •́)y

接下来是 bonus flag
其实 bonus flag 的反混淆部分和正常的差不多，只是要找到反反编译的地方：
目标在这里，也是一个 call $+5 and retn 的操作，怎么找到的呢？



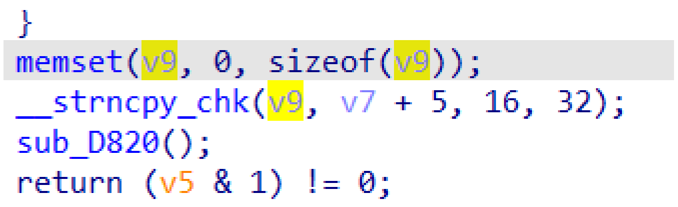
```
text:0000DCC2                 mov     byte ptr [esp+57h], 1
text:0000DCC7                 jmp     loc_DE29
text:0000DCCC ; ---------------------------------------------------------------------------
text:0000DCCC
text:0000DCCC loc_DCCC:                               ; CODE XREF: Java_com_pore_lab10_1task_MainActivity_Check+2FD↑j
text:0000DCCC                                         ; Java_com_pore_lab10_1task_MainActivity_Check+31D↑j
text:0000DCCC                 nop                     ; Keypatch filled range [0xDCCC:0xDCD7] (12 bytes), replaced:
text:0000DCCC                                         ;   call $+5
text:0000DCCC                                         ;   add dword ptr [esp], 7
text:0000DCCC                                         ;   xor eax, eax
text:0000DCCC                                         ;   retn
text:0000DCCD                 nop
text:0000DCCE                 nop
text:0000DCCF                 nop
text:0000DCD0                 nop
text:0000DCD1                 nop
text:0000DCD2                 nop
text:0000DCD3                 nop
text:0000DCD4                 nop
text:0000DCD5                 nop
text:0000DCD6                 nop
text:0000DCD7                 nop
text:0000DCD8                 mov     eax, 0FFFFFFFFh
text:0000DCDD                 lea     ecx, [esp+10Ch+var_54]
```

我大概猜测是通过一些方式把另外一个 retn 藏起来了，于是把原来已经自动反编译的主函数在回去看，找到了一个没用的函数



```
.text:0000DDFA
.text:0000DDFA loc_DDFA:                               ; CODE XREF: Java_com_pore_lab10_1task_MainActivity_Check+40F↑j
.text:0000DDFA                 lea     eax, [esp+10Ch+var_54]
.text:0000DE01                 mov     [esp], eax      ; char *
.text:0000DE04                 mov     ebx, [esp+4Ch]
.text:0000DE08                 call    __Z8sub_D820Pc  ; sub_D820(char *)
.text:0000DE0D                 test    al, 1
.text:0000DE0F                 jnz     loc_DE1A
.text:0000DE15                 jmp     loc_DE24
.text:0000DE1A ; ---------------------------------------------------------------------------
.text:0000DE1A
.text:0000DE1A loc_DE1A:                               ; CODE XREF: Java_com_pore_lab10_1task_MainActivity_Check+46F↑j
.text:0000DE1A                 mov     byte ptr [esp+57h], 1
.text:0000DE1F                 jmp     loc_DE29
```

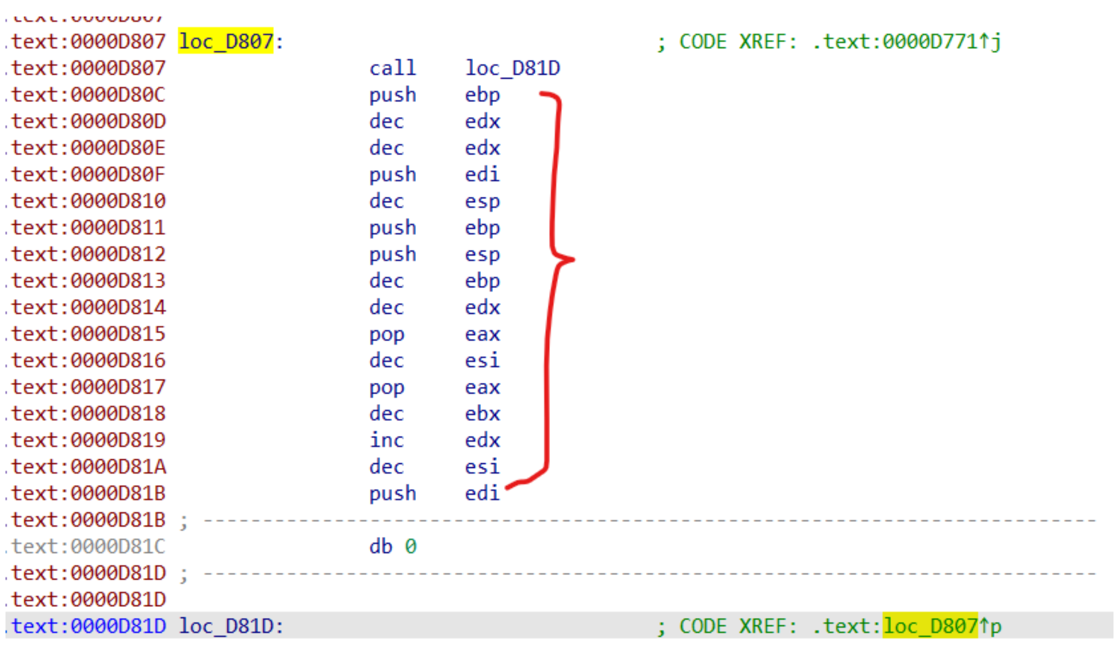
那基本上能确定，这个函数没有被反编译出来，也就是上面的 call + retn 把这个藏起来了，把它通过 keypatch nop 掉后露出来了



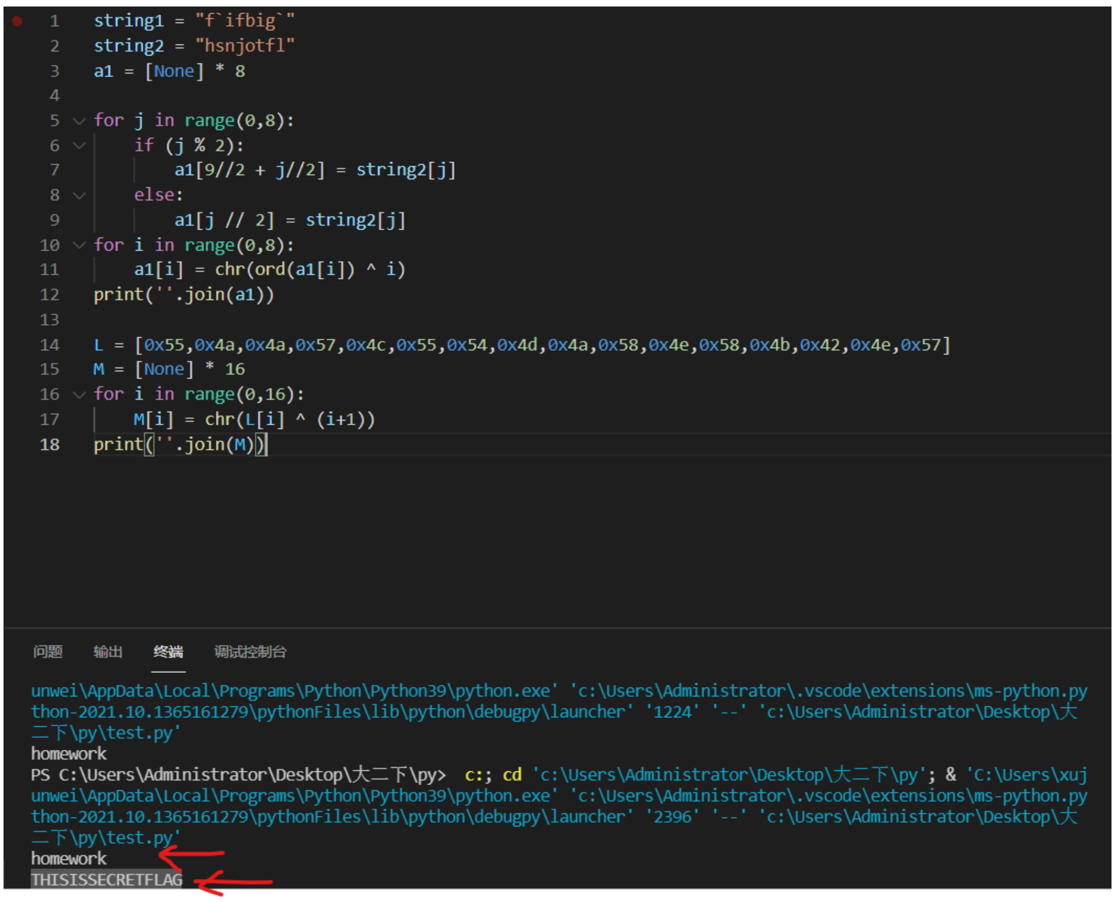
```
}
memset(v9, 0, sizeof(v9));
__strncpy_chk(v9, v7 + 5, 16, 32);
sub_D820();
return (v5 & 1) != 0;
```

ok，找到入口了，剩下的反反汇编的 skills 基本和之前的一致，这里就不多阐述了吧（和第一个 flag 的第一部分几乎一模一样）

下面这一段最后 strcmp 的数据段（这里还没转数据），call 指令把这里的地址放入栈，最后比较字符串就是这个地址



```
.text:0000D807 loc_D807:                               ; CODE XREF: .text:0000D771↑j
.text:0000D807                 call    loc_D81D
.text:0000D80C                 push    ebp
.text:0000D80D                 dec     edx
.text:0000D80E                 dec     edx
.text:0000D80F                 push    edi
.text:0000D810                 dec     esp
.text:0000D811                 push    ebp
.text:0000D812                 push    esp
.text:0000D813                 dec     ebp
.text:0000D814                 dec     edx
.text:0000D815                 pop     eax
.text:0000D816                 dec     esi
.text:0000D817                 pop     eax
.text:0000D818                 dec     ebx
.text:0000D819                 inc     edx
.text:0000D81A                 dec     esi
.text:0000D81B                 push    edi
.text:0000D81B ; ---------------------------------------------------------------------------
.text:0000D81C                 db 0
.text:0000D81D ; ---------------------------------------------------------------------------
.text:0000D81D
.text:0000D81D loc_D81D:                               ; CODE XREF: .text:loc_D807↑p
```

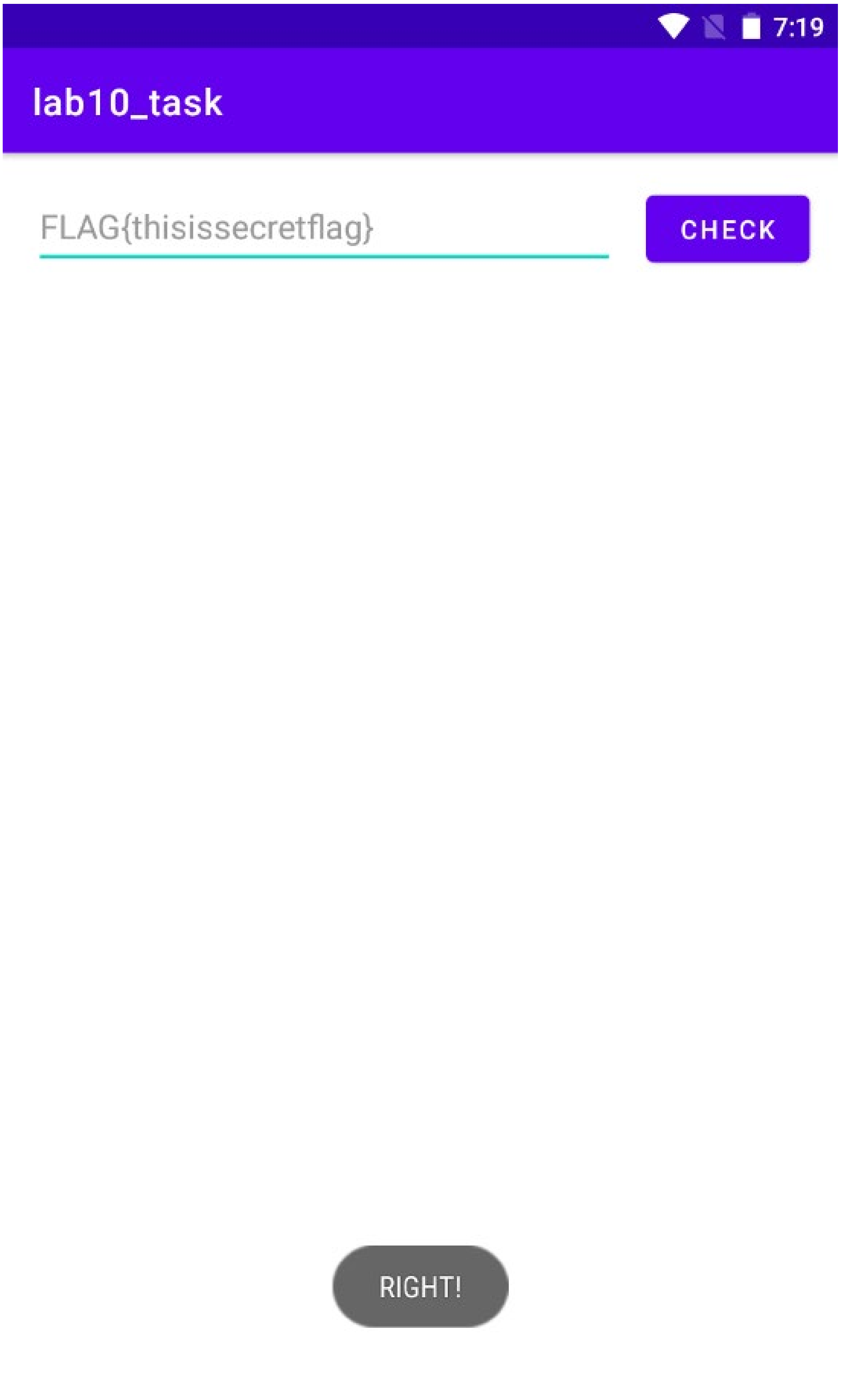


```python
string1 = "f`ifbig`"
string2 = "hsnjotfl"
a1 = [None] * 8

for j in range(0,8):
    if (j % 2):
        a1[9//2 + j//2] = string2[j]
    else:
        a1[j // 2] = string2[j]
for i in range(0,8):
    a1[i] = chr(ord(a1[i]) ^ i)
print(''.join(a1))

L = [0x55,0x4a,0x4a,0x57,0x4c,0x55,0x54,0x4d,0x4a,0x58,0x4e,0x58,0x4b,0x42,0x4e,0x57]
M = [None] * 16
for i in range(0,16):
    M[i] = chr(L[i] ^ (i+1))
print(''.join(M))
```

```
unwei\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.py
thon-2021.10.1365161279\pythonFiles\lib\python\debugpy\launcher' '1224' '--' 'c:\Users\Administrator\Desktop\大
二下\py\test.py'
homework
PS C:\Users\Administrator\Desktop\大二下\py>  c:; cd 'c:\Users\Administrator\Desktop\大二下\py'; & 'C:\Users\xuj
unwei\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.py
thon-2021.10.1365161279\pythonFiles\lib\python\debugpy\launcher' '2396' '--' 'c:\Users\Administrator\Desktop\大
二下\py\test.py'
homework
THISISSECRETFLAG
```

指的是第一个和第二个 flag

当然最后的答案是小写的……



FLAG{thisissecretflag}

RIGHT!