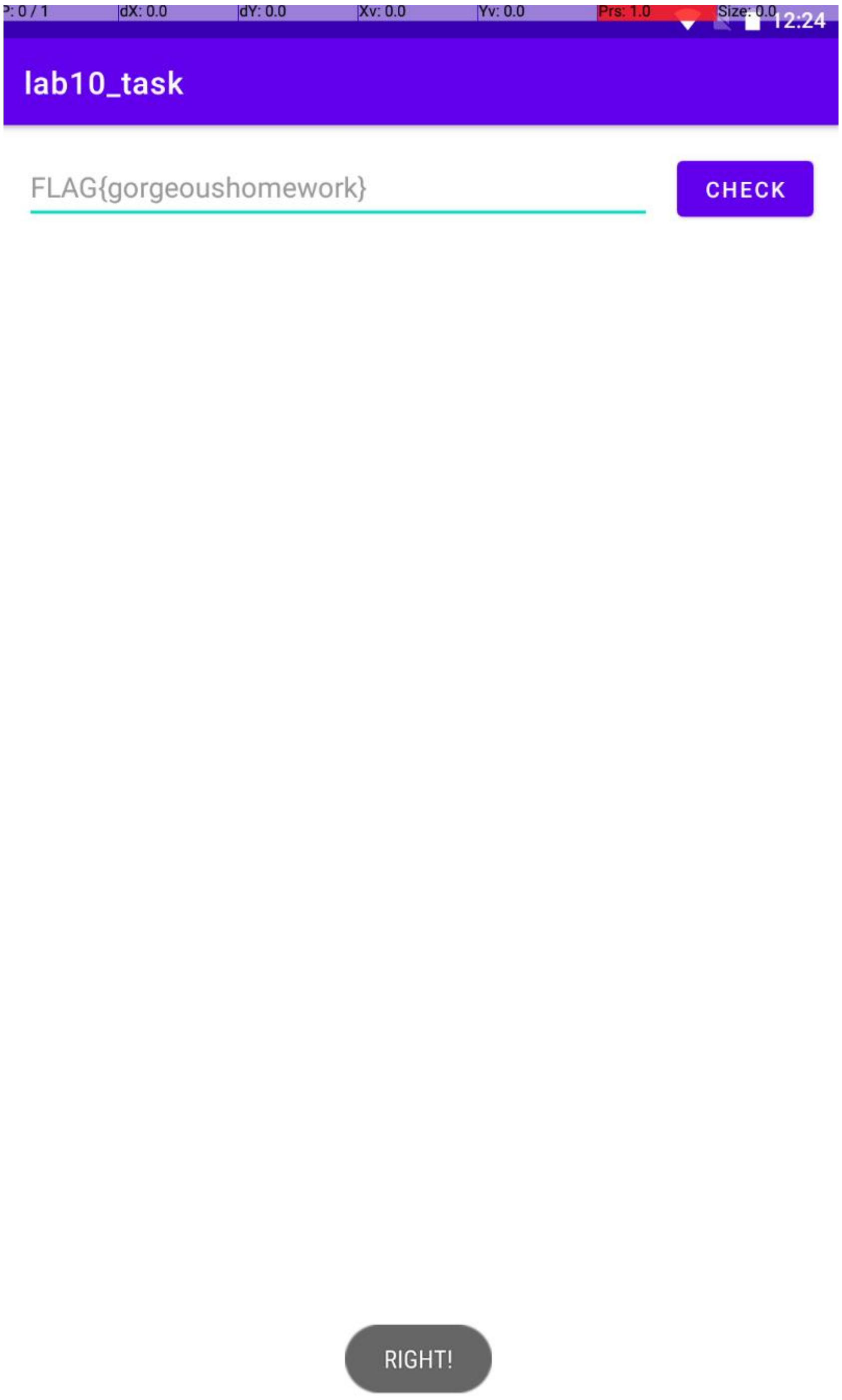
Lab10 Advanced Reverse

STU ID: 20307130350

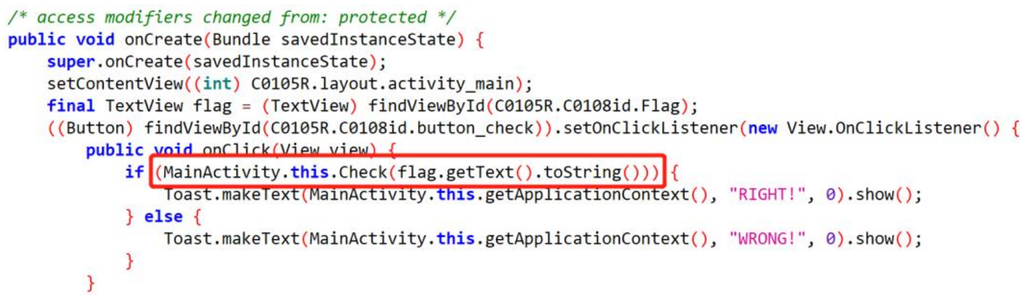
Your Flag: FLAG{gorgeoushomework}

Bonus Flag：未找到



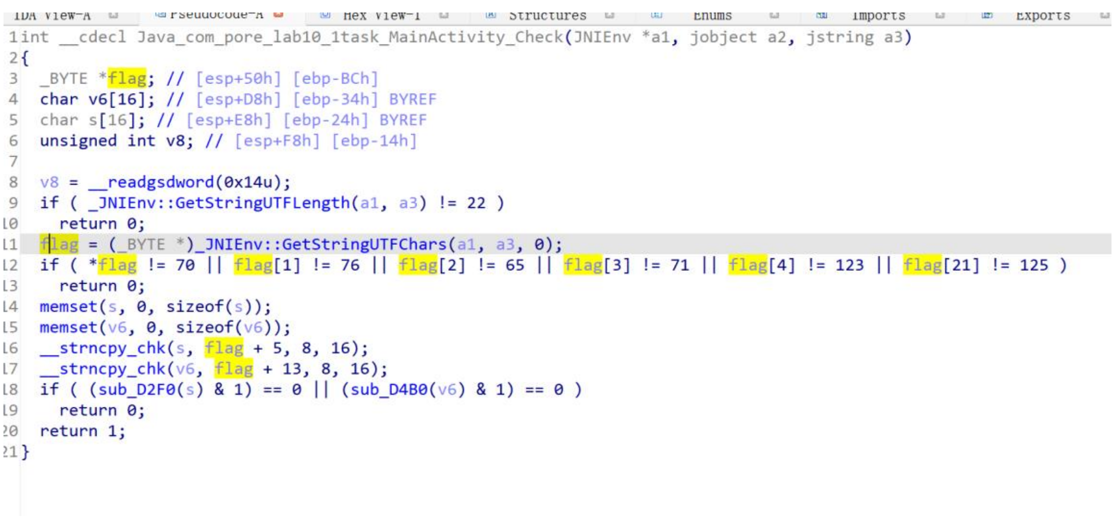
Analysis Process Breakdown:

1. 用 jadx 打开 lab10.apk 之后搜索 WRONG，可以直接定位到我们需要的函数，即 native check 函数。

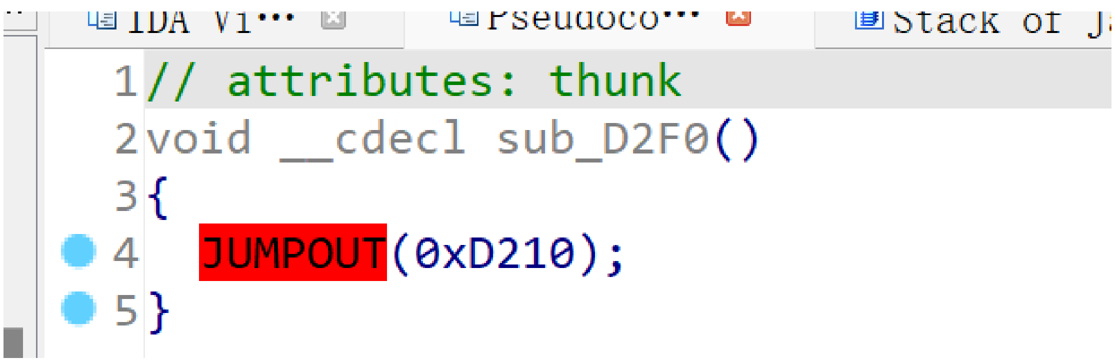


```java
/* access modifiers changed from: protected */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView((int) C0105R.layout.activity_main);
    final TextView flag = (TextView) findViewById(C0105R.C0108id.Flag);
    ((Button) findViewById(C0105R.C0108id.button_check)).setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            if (MainActivity.this.Check(flag.getText().toString())) {
                Toast.makeText(MainActivity.this.getApplicationContext(), "RIGHT!", 0).show();
            } else {
                Toast.makeText(MainActivity.this.getApplicationContext(), "WRONG!", 0).show();
            }
        }
```

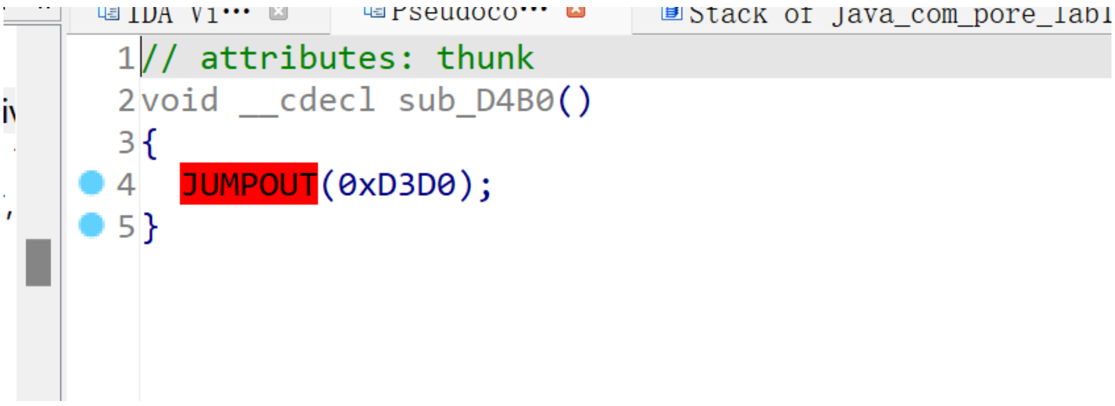
2. 用 ida 打开 so 文件，从 check 函数的伪代码可以清楚地知道 flag 的长度是 22，且中间的 16 个字符被分成了两部分，分别用两个 char[8]数组来接收，分别用 sub_D2F0 和 sub_D4B0 来检验。且这两个函数是 jumpout，需要 patch。



```c
int __cdecl Java_com_pore_lab10_1task_MainActivity_Check(JNIEnv *a1, jobject a2, jstring a3)
{
  _BYTE *flag; // [esp+50h] [ebp-BCh]
  char v6[16]; // [esp+D8h] [ebp-34h] BYREF
  char s[16]; // [esp+E8h] [ebp-24h] BYREF
  unsigned int v8; // [esp+F8h] [ebp-14h]

  v8 = __readgsdword(0x14u);
  if ( _JNIEnv::GetStringUTFLength(a1, a3) != 22 )
    return 0;
  flag = (_BYTE *)_JNIEnv::GetStringUTFChars(a1, a3, 0);
  if ( *flag != 70 || flag[1] != 76 || flag[2] != 65 || flag[3] != 71 || flag[4] != 123 || flag[21] != 125 )
    return 0;
  memset(s, 0, sizeof(s));
  memset(v6, 0, sizeof(v6));
  __strncpy_chk(s, flag + 5, 8, 16);
  __strncpy_chk(v6, flag + 13, 8, 16);
  if ( (sub_D2F0(s) & 1) == 0 || (sub_D4B0(v6) & 1) == 0 )
    return 0;
  return 1;
}
```

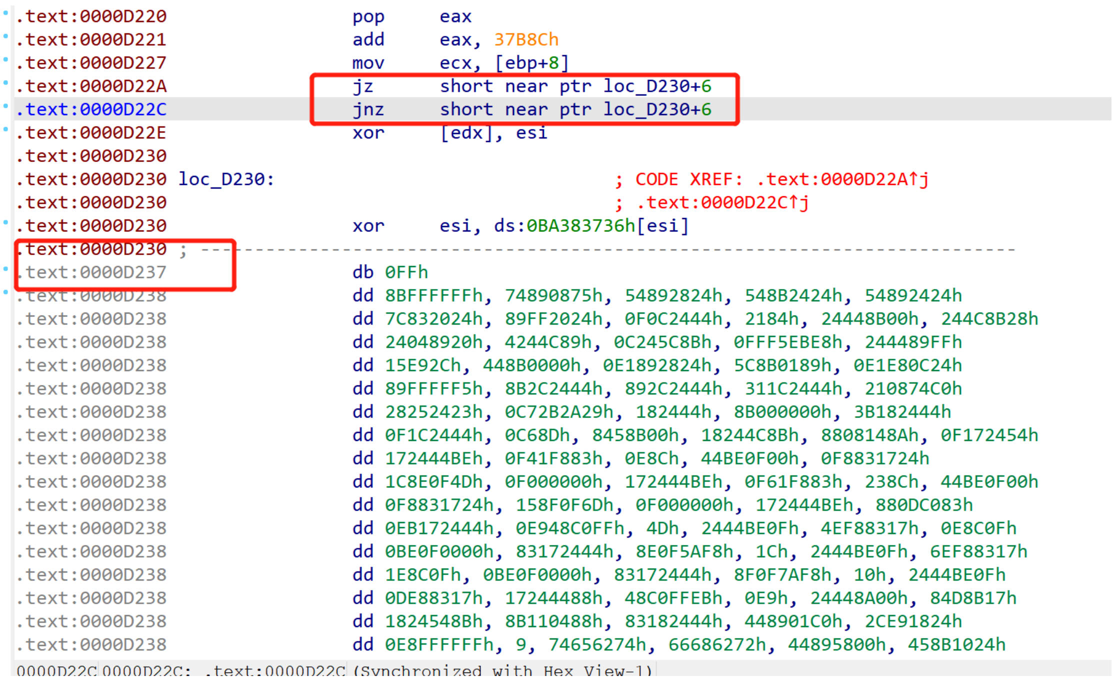


```c
// attributes: thunk
void __cdecl sub_D2F0()
{
  JUMPOUT(0xD210);
}
```

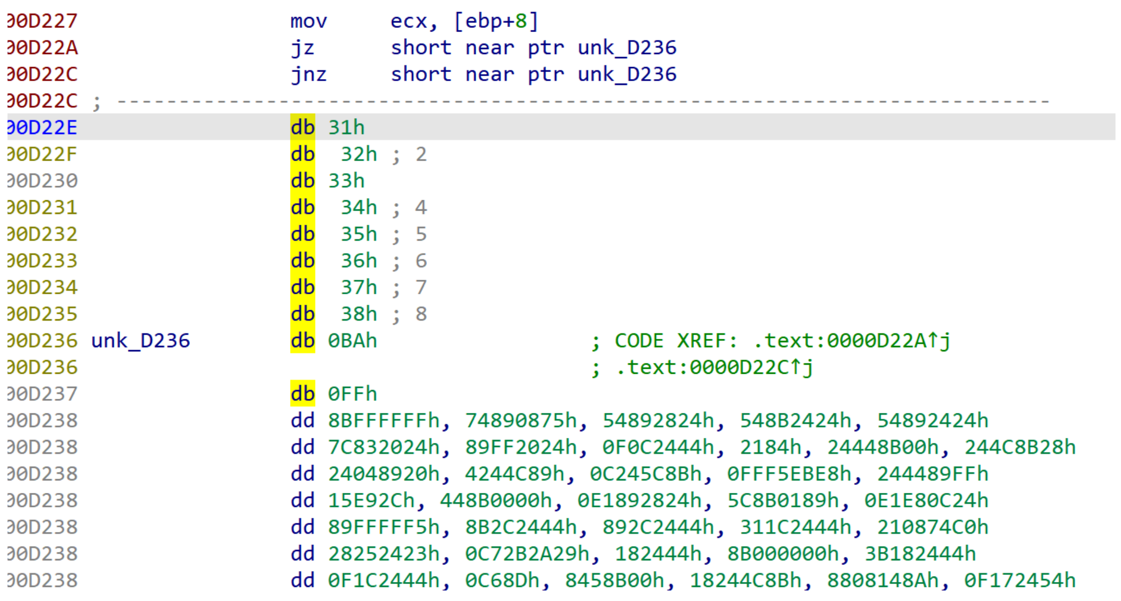


```
// attributes: thunk
void __cdecl sub_D4B0()
{
  JUMPOUT(0xD3D0);
}
```

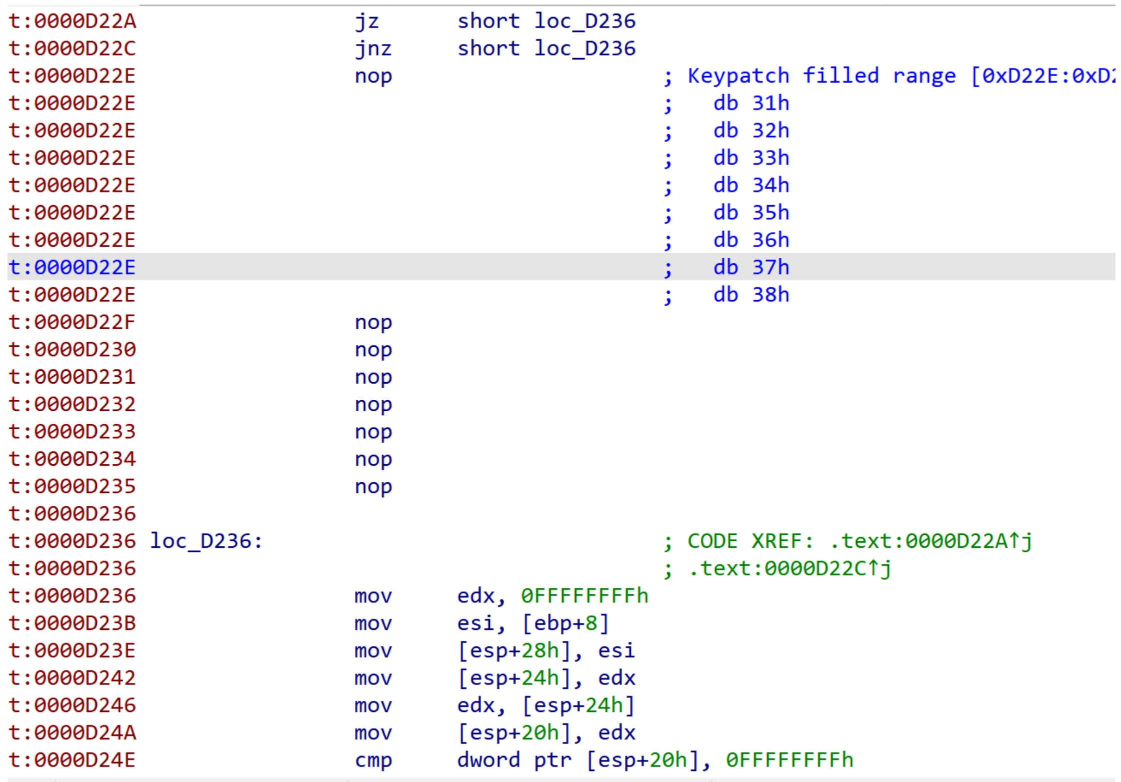
3. 对于 sub_D2F0
   a) 首先是 Jump Instr,with the Same Target 的花指令，其 target 处的地址没有指令，且后面是 data，可见 0000D22E~0000D230 是花指令，因为目标地址的字节在花指令的字节中，所以先将其转换为 data，将无用的数据变成 nop 即可还原。



```
.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
.text:0000D238                 dd 0EB172444h, 0E948C0FFh, 4Dh, 2444BE0Fh, 4EF88317h, 0E8C0Fh
.text:0000D238                 dd 0BE0F0000h, 83172444h, 8E0F5AF8h, 1Ch, 2444BE0Fh, 6EF88317h
.text:0000D238                 dd 1E8C0Fh, 0BE0F0000h, 83172444h, 8F0F7AF8h, 10h, 2444BE0Fh
.text:0000D238                 dd 0DE88317h, 17244488h, 48C0FFEBh, 0E9h, 24448A00h, 84D8B17h
.text:0000D238                 dd 1824548Bh, 8B110488h, 83182444h, 448901C0h, 2CE91824h
.text:0000D238                 dd 0E8FFFFFFh, 9, 74656274h, 66686272h, 44895800h, 458B1024h
0000D22C 0000D22C: .text:0000D22C (Synchronized with Hex View-1)
```

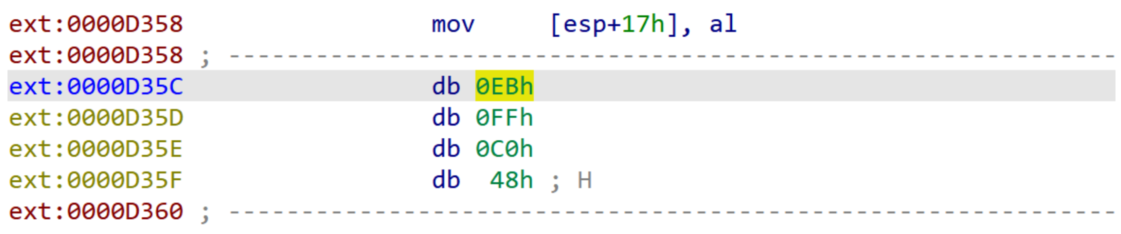


```
00D227                 mov     ecx, [ebp+8]
00D22A                 jz      short near ptr unk_D236
00D22C                 jnz     short near ptr unk_D236
00D22C ; ---------------------------------------------------------------------------
00D22E                 db 31h
00D22F                 db  32h ; 2
00D230                 db 33h
00D231                 db  34h ; 4
00D232                 db  35h ; 5
00D233                 db  36h ; 6
00D234                 db  37h ; 7
00D235                 db  38h ; 8
00D236 unk_D236        db 0BAh                 ; CODE XREF: .text:0000D22A↑j
00D236                                         ; .text:0000D22C↑j
00D237                 db 0FFh
00D238                 dd 8BFFFFFFh, 74890875h, 54892824h, 548B2424h, 54892424h
00D238                 dd 7C832024h, 89FF2024h, 0F0C2444h, 2184h, 24448B00h, 244C8B28h
00D238                 dd 24048920h, 4244C89h, 0C245C8Bh, 0FFF5EBE8h, 244489FFh
00D238                 dd 15E92Ch, 448B0000h, 0E1892824h, 5C8B0189h, 0E1E80C24h
00D238                 dd 89FFFFF5h, 8B2C2444h, 892C2444h, 311C2444h, 210874C0h
00D238                 dd 28252423h, 0C72B2A29h, 182444h, 8B000000h, 3B182444h
00D238                 dd 0F1C2444h, 0C68Dh, 8458B00h, 18244C8Bh, 8808148Ah, 0F172454h
```

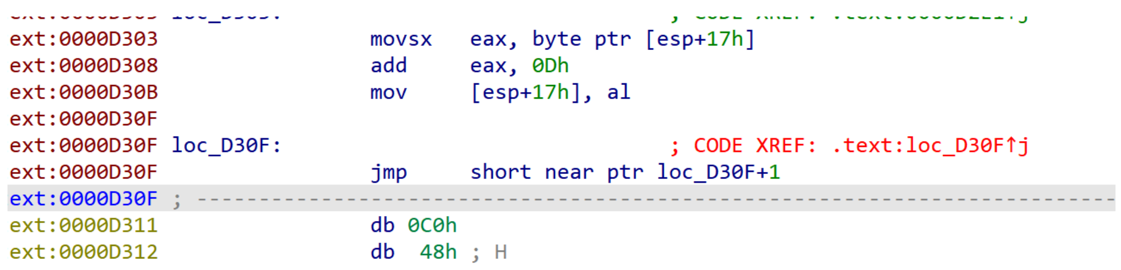


```
t:0000D22A                 jz      short loc_D236
t:0000D22C                 jnz     short loc_D236
t:0000D22E                 nop                          ; Keypatch filled range [0xD22E:0xD
t:0000D22E                                              ;   db 31h
t:0000D22E                                              ;   db 32h
t:0000D22E                                              ;   db 33h
t:0000D22E                                              ;   db 34h
t:0000D22E                                              ;   db 35h
t:0000D22E                                              ;   db 36h
t:0000D22E                                              ;   db 37h
t:0000D22E                                              ;   db 38h
t:0000D22F                 nop
t:0000D230                 nop
t:0000D231                 nop
t:0000D232                 nop
t:0000D233                 nop
t:0000D234                 nop
t:0000D235                 nop
t:0000D236
t:0000D236 loc_D236:                                    ; CODE XREF: .text:0000D22A↑j
t:0000D236                                              ; .text:0000D22C↑j
t:0000D236                 mov     edx, 0FFFFFFFFh
t:0000D23B                 mov     esi, [ebp+8]
t:0000D23E                 mov     [esp+28h], esi
t:0000D242                 mov     [esp+24h], edx
t:0000D246                 mov     edx, [esp+24h]
t:0000D24A                 mov     [esp+20h], edx
t:0000D24E                 cmp     dword ptr [esp+20h], 0FFFFFFFFh
```

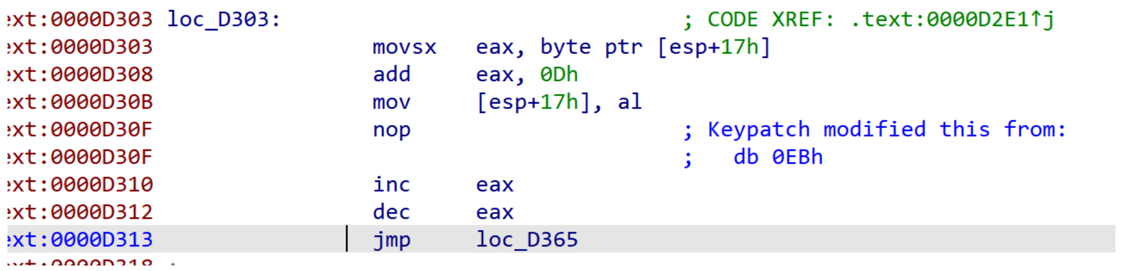
B）其次是三个 Impossible Disassembly 类型的花指令，即一份字节为两个汇编指令的组成部分，因此可以 data jmp 指令，将 0EBh(即 jmp) nop 之后变成代码即可。



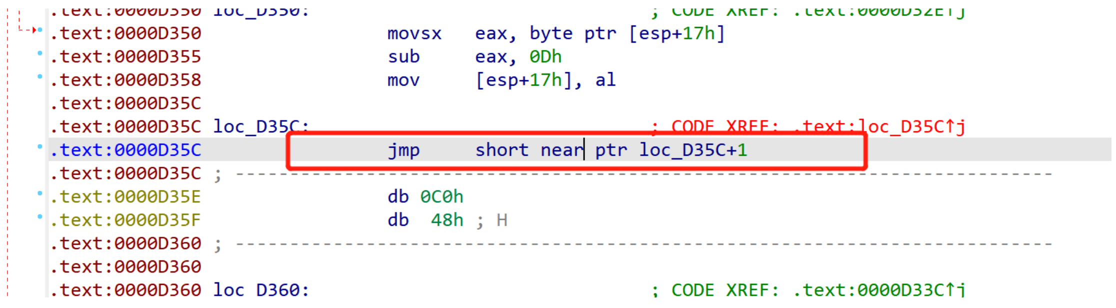
```
ext:0000D358                 mov     [esp+17h], al
ext:0000D358 ; ---------------------------------------------------------------
ext:0000D35C                 db 0EBh
ext:0000D35D                 db 0FFh
ext:0000D35E                 db 0C0h
ext:0000D35F                 db  48h ; H
ext:0000D360 ; ---------------------------------------------------------------
```



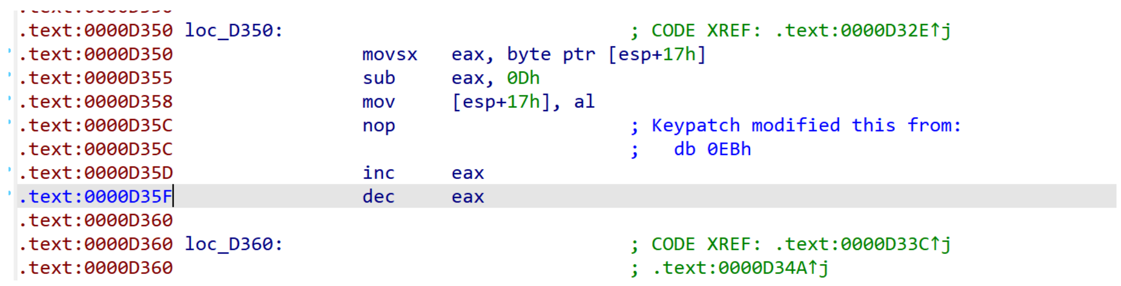
```
ext:0000D303                movsx   eax, byte ptr [esp+17h]
ext:0000D308                add     eax, 0Dh
ext:0000D30B                mov     [esp+17h], al
ext:0000D30F
ext:0000D30F loc_D30F:                              ; CODE XREF: .text:loc_D30F↑j
ext:0000D30F                jmp     short near ptr loc_D30F+1
ext:0000D30F ; ---------------------------------------------------------------------------
ext:0000D311                db 0C0h
ext:0000D312                db  48h ; H
```



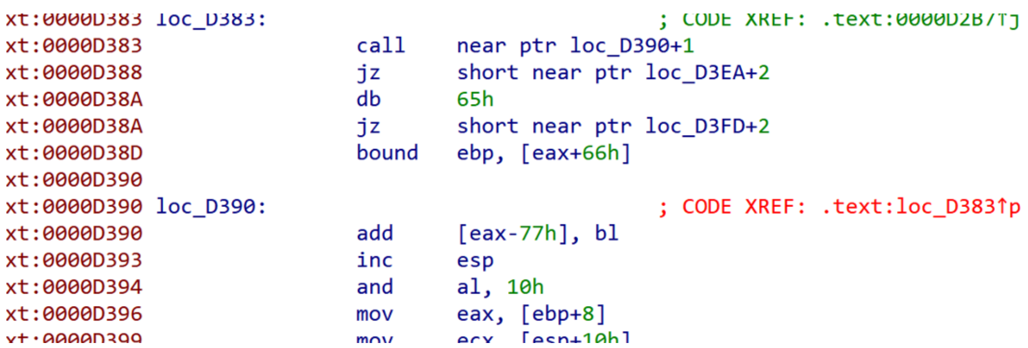
```
xt:0000D303 loc_D303:                              ; CODE XREF: .text:0000D2E1↑j
xt:0000D303                movsx   eax, byte ptr [esp+17h]
xt:0000D308                add     eax, 0Dh
xt:0000D30B                mov     [esp+17h], al
xt:0000D30F                nop                     ; Keypatch modified this from:
xt:0000D30F                                        ;   db 0EBh
xt:0000D310                inc     eax
xt:0000D312                dec     eax
xt:0000D313                jmp     loc_D365
```



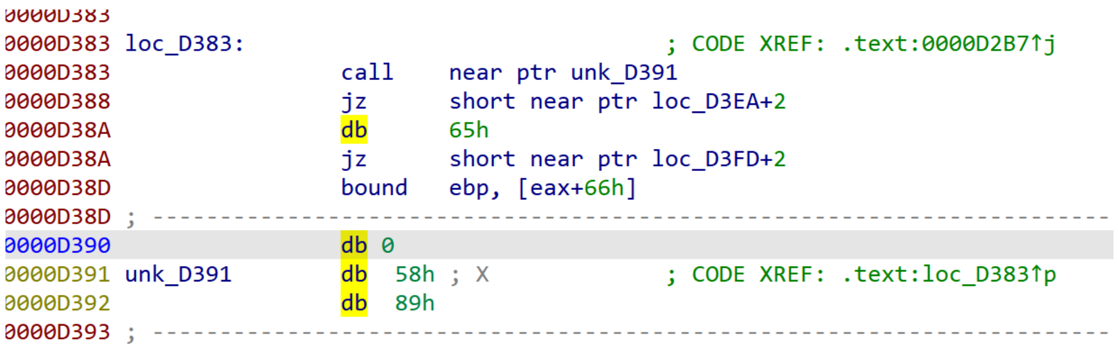
```
.text:0000D350                movsx   eax, byte ptr [esp+17h]
.text:0000D355                sub     eax, 0Dh
.text:0000D358                mov     [esp+17h], al
.text:0000D35C
.text:0000D35C loc_D35C:                              ; CODE XREF: .text:loc_D35C↑j
.text:0000D35C                jmp     short near ptr loc_D35C+1
.text:0000D35C ; ---------------------------------------------------------------------------
.text:0000D35E                db 0C0h
.text:0000D35F                db  48h ; H
.text:0000D360 ; ---------------------------------------------------------------------------
.text:0000D360
.text:0000D360 loc_D360:                              ; CODE XREF: .text:0000D33C↑j
```



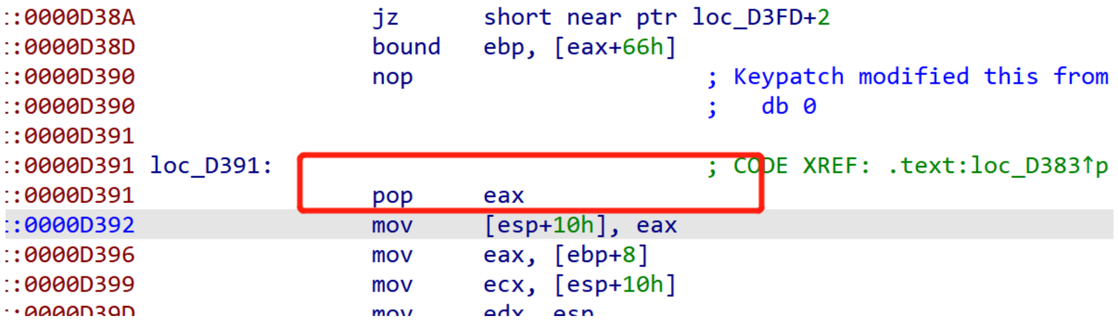
```
.text:0000D350 loc_D350:                              ; CODE XREF: .text:0000D32E↑j
.text:0000D350                movsx   eax, byte ptr [esp+17h]
.text:0000D355                sub     eax, 0Dh
.text:0000D358                mov     [esp+17h], al
.text:0000D35C                nop                     ; Keypatch modified this from:
.text:0000D35C                                        ;   db 0EBh
.text:0000D35D                inc     eax
.text:0000D35F                dec     eax
.text:0000D360
.text:0000D360 loc_D360:                              ; CODE XREF: .text:0000D33C↑j
.text:0000D360                                        ; .text:0000D34A↑j
```



```
xt:0000D383                call    near ptr loc_D390+1
xt:0000D388                jz      short near ptr loc_D3EA+2
xt:0000D38A                db      65h
xt:0000D38A                jz      short near ptr loc_D3FD+2
xt:0000D38D                bound   ebp, [eax+66h]
xt:0000D390
xt:0000D390 loc_D390:                              ; CODE XREF: .text:loc_D383↑p
xt:0000D390                add     [eax-77h], bl
xt:0000D393                inc     esp
xt:0000D394                and     al, 10h
xt:0000D396                mov     eax, [ebp+8]
```

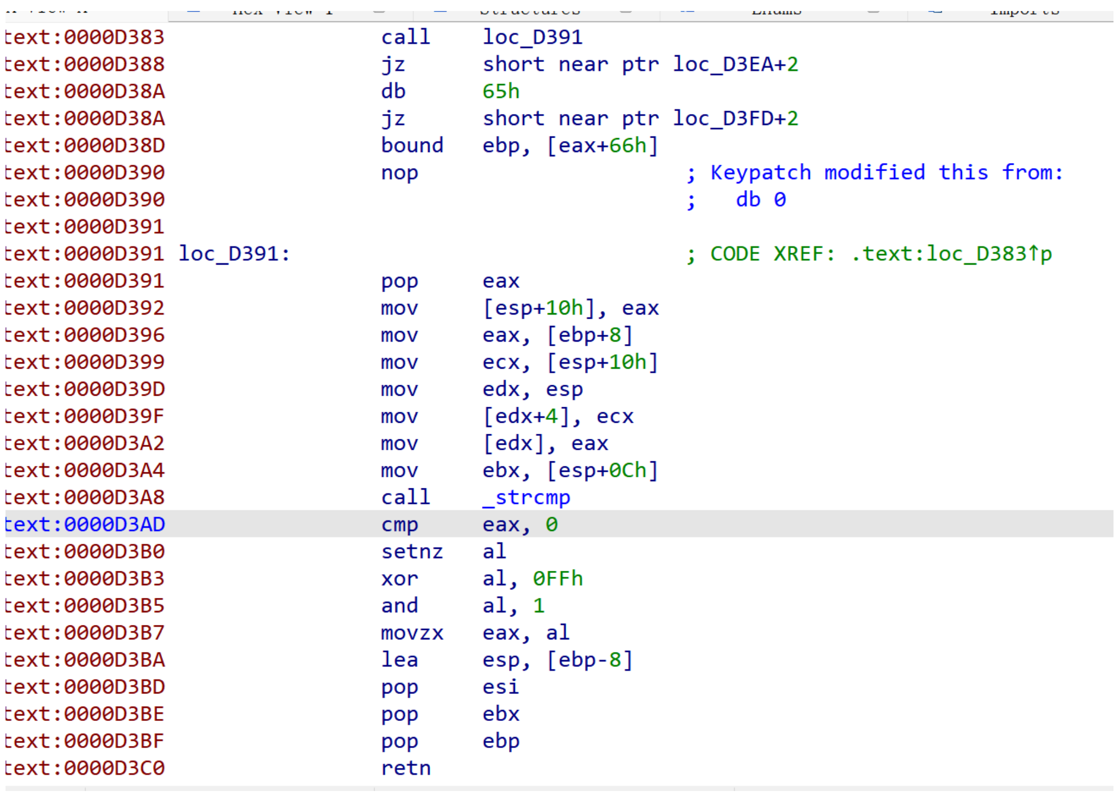


```
0000D383 loc_D383:                                   ; CODE XREF: .text:0000D2B7↑j
0000D383                call    near ptr unk_D391
0000D388                jz      short near ptr loc_D3EA+2
0000D38A                db      65h
0000D38A                jz      short near ptr loc_D3FD+2
0000D38D                bound   ebp, [eax+66h]
0000D38D ; ---------------------------------------------------------------------------
0000D390                db 0
0000D391 unk_D391       db  58h ; X                  ; CODE XREF: .text:loc_D383↑p
0000D392                db  89h
0000D393 ; ---------------------------------------------------------------------------
```

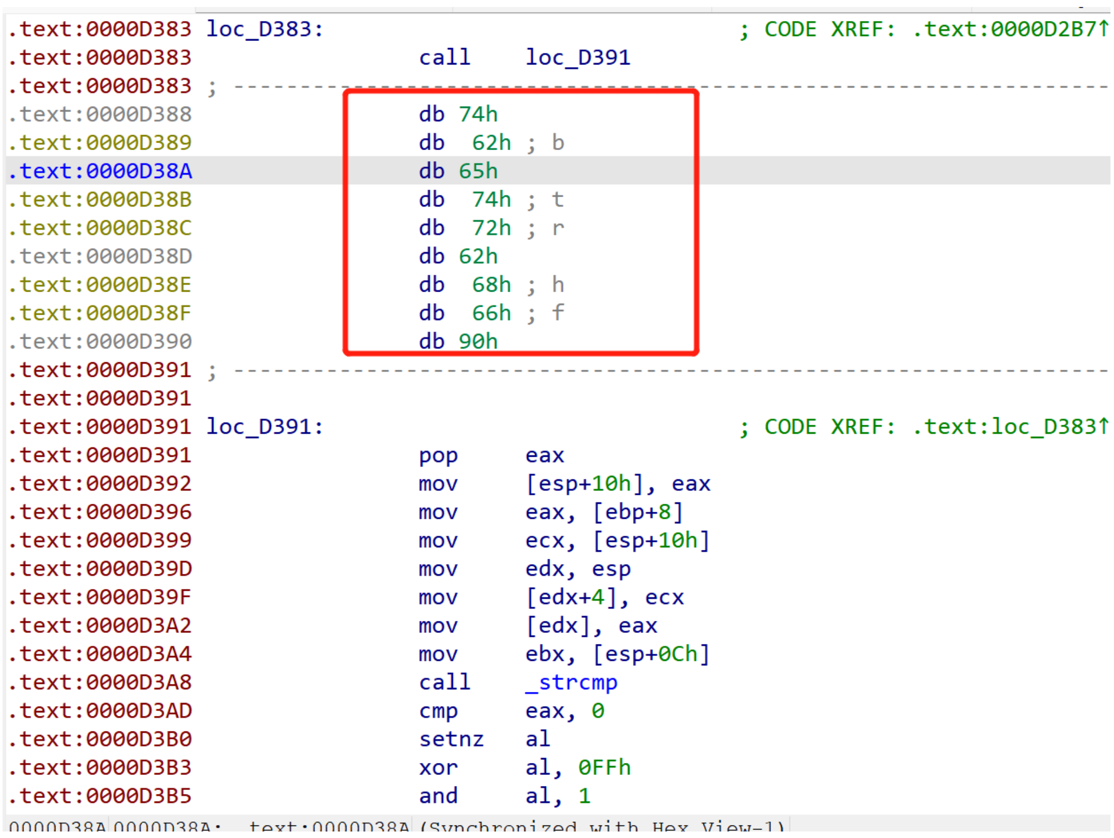


```
:0000D38A               jz      short near ptr loc_D3FD+2
:0000D38D               bound   ebp, [eax+66h]
:0000D390               nop                          ; Keypatch modified this from
:0000D390                                            ;   db 0
:0000D391
:0000D391 loc_D391:                                  ; CODE XREF: .text:loc_D383↑p
:0000D391               pop     eax
:0000D392               mov     [esp+10h], eax
:0000D396               mov     eax, [ebp+8]
:0000D399               mov     ecx, [esp+10h]
```

C）最后是 call-retn 问题，在 call 之后的下一步地址压栈，根据 sub_D391 的逻辑，它其实是 strcmp 函数。根据堆栈的信息，得到 strcmp 的参数的地址。

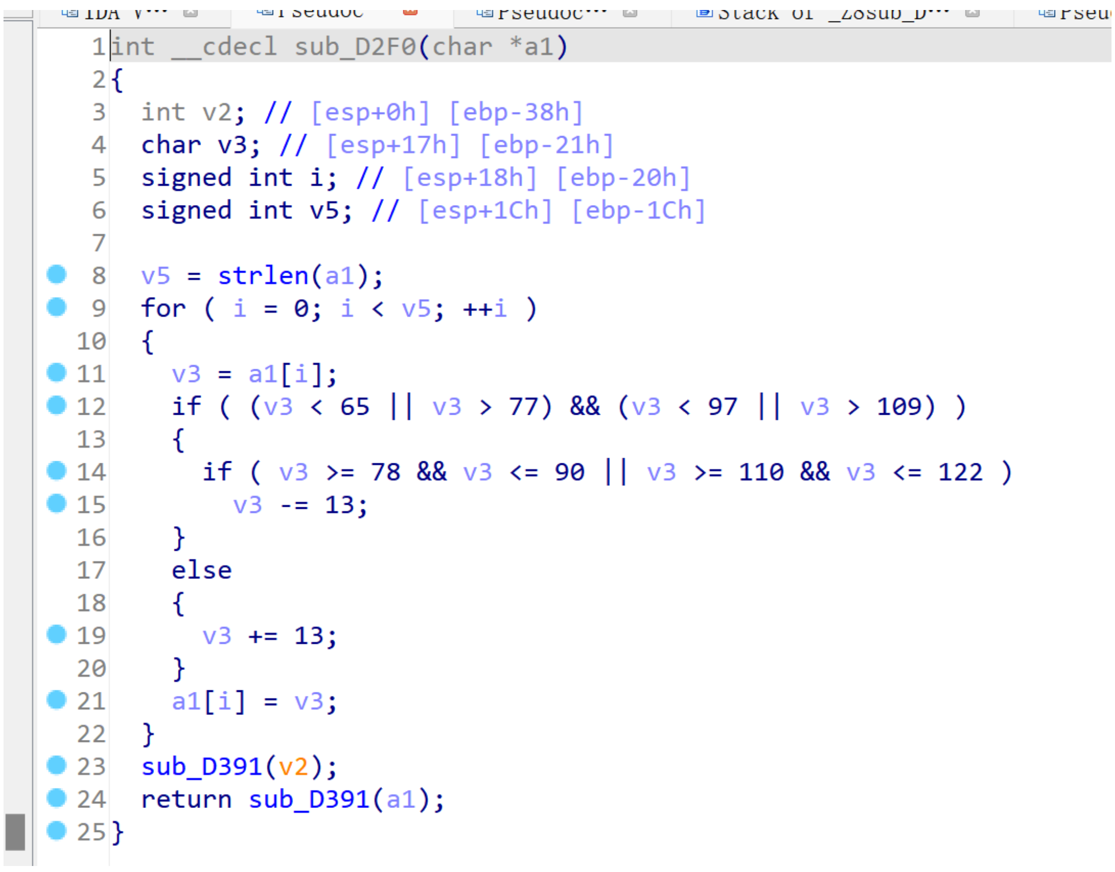


```
text:0000D383                   call    loc_D391
text:0000D388                   jz      short near ptr loc_D3EA+2
text:0000D38A                   db      65h
text:0000D38A                   jz      short near ptr loc_D3FD+2
text:0000D38D                   bound   ebp, [eax+66h]
text:0000D390                   nop                     ; Keypatch modified this from:
text:0000D390                                           ;   db 0
text:0000D391
text:0000D391 loc_D391:                                 ; CODE XREF: .text:loc_D383↑p
text:0000D391                   pop     eax
text:0000D392                   mov     [esp+10h], eax
text:0000D396                   mov     eax, [ebp+8]
text:0000D399                   mov     ecx, [esp+10h]
text:0000D39D                   mov     edx, esp
text:0000D39F                   mov     [edx+4], ecx
text:0000D3A2                   mov     [edx], eax
text:0000D3A4                   mov     ebx, [esp+0Ch]
text:0000D3A8                   call    _strcmp
text:0000D3AD                   cmp     eax, 0
text:0000D3B0                   setnz   al
text:0000D3B3                   xor     al, 0FFh
text:0000D3B5                   and     al, 1
text:0000D3B7                   movzx   eax, al
text:0000D3BA                   lea     esp, [ebp-8]
text:0000D3BD                   pop     esi
text:0000D3BE                   pop     ebx
text:0000D3BF                   pop     ebp
text:0000D3C0                   retn
```



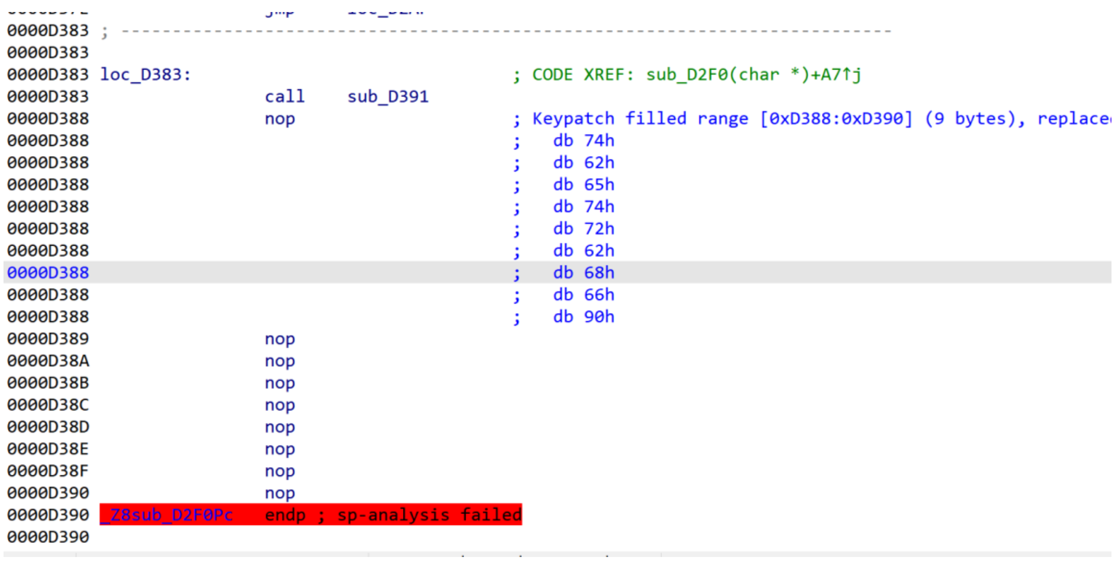
```
.text:0000D383 loc_D383:                               ; CODE XREF: .text:0000D2B7↑
.text:0000D383                 call    loc_D391
.text:0000D383 ; ---------------------------------------------------------------------------
.text:0000D388                 db 74h
.text:0000D389                 db  62h ; b
.text:0000D38A                 db 65h
.text:0000D38B                 db  74h ; t
.text:0000D38C                 db  72h ; r
.text:0000D38D                 db 62h
.text:0000D38E                 db  68h ; h
.text:0000D38F                 db  66h ; f
.text:0000D390                 db 90h
.text:0000D391 ; ---------------------------------------------------------------------------
.text:0000D391
.text:0000D391 loc_D391:                               ; CODE XREF: .text:loc_D383↑
.text:0000D391                 pop     eax
.text:0000D392                 mov     [esp+10h], eax
.text:0000D396                 mov     eax, [ebp+8]
.text:0000D399                 mov     ecx, [esp+10h]
.text:0000D39D                 mov     edx, esp
.text:0000D39F                 mov     [edx+4], ecx
.text:0000D3A2                 mov     [edx], eax
.text:0000D3A4                 mov     ebx, [esp+0Ch]
.text:0000D3A8                 call    _strcmp
.text:0000D3AD                 cmp     eax, 0
.text:0000D3B0                 setnz   al
.text:0000D3B3                 xor     al, 0FFh
.text:0000D3B5                 and     al, 1
0000D38A 0000D38A: .text:0000D38A (Synchronized with Hex View-1)
```

F5 时需要将 db 部分 nop 掉，否则会出错。



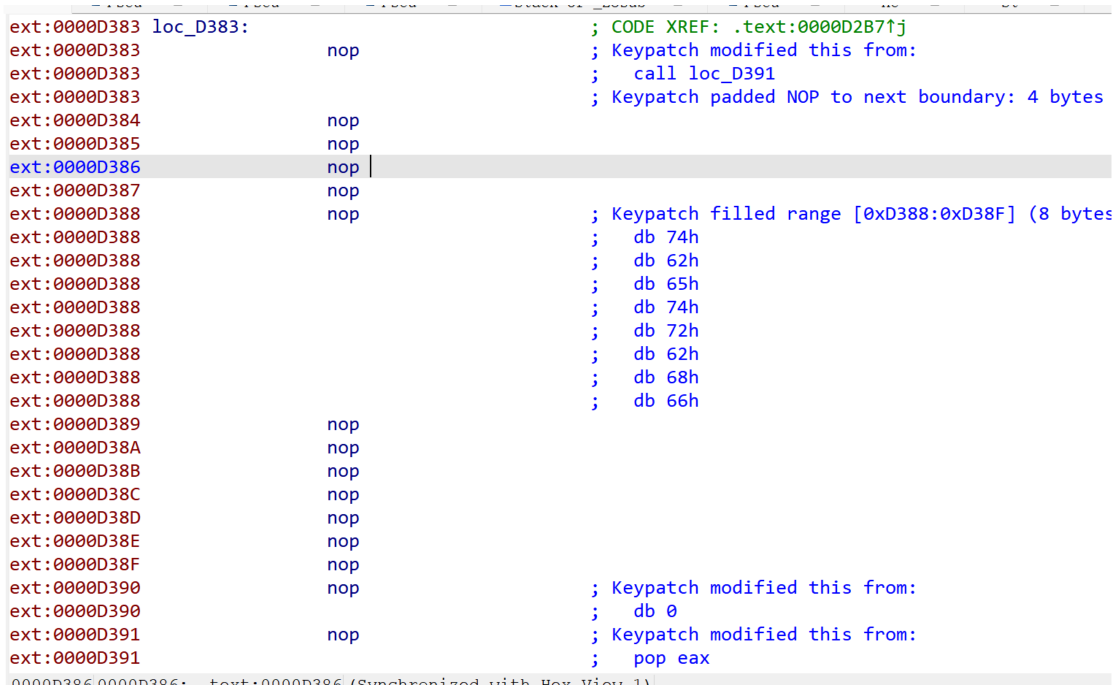
```
int __cdecl sub_D2F0(char *a1)
{
  int v2; // [esp+0h] [ebp-38h]
  char v3; // [esp+17h] [ebp-21h]
  signed int i; // [esp+18h] [ebp-20h]
  signed int v5; // [esp+1Ch] [ebp-1Ch]

  v5 = strlen(a1);
  for ( i = 0; i < v5; ++i )
  {
    v3 = a1[i];
    if ( (v3 < 65 || v3 > 77) && (v3 < 97 || v3 > 109) )
    {
      if ( v3 >= 78 && v3 <= 90 || v3 >= 110 && v3 <= 122 )
        v3 -= 13;
    }
    else
    {
      v3 += 13;
    }
    a1[i] = v3;
  }
  sub_D391(v2);
  return sub_D391(a1);
}
```

F5 之后汇编代码变成

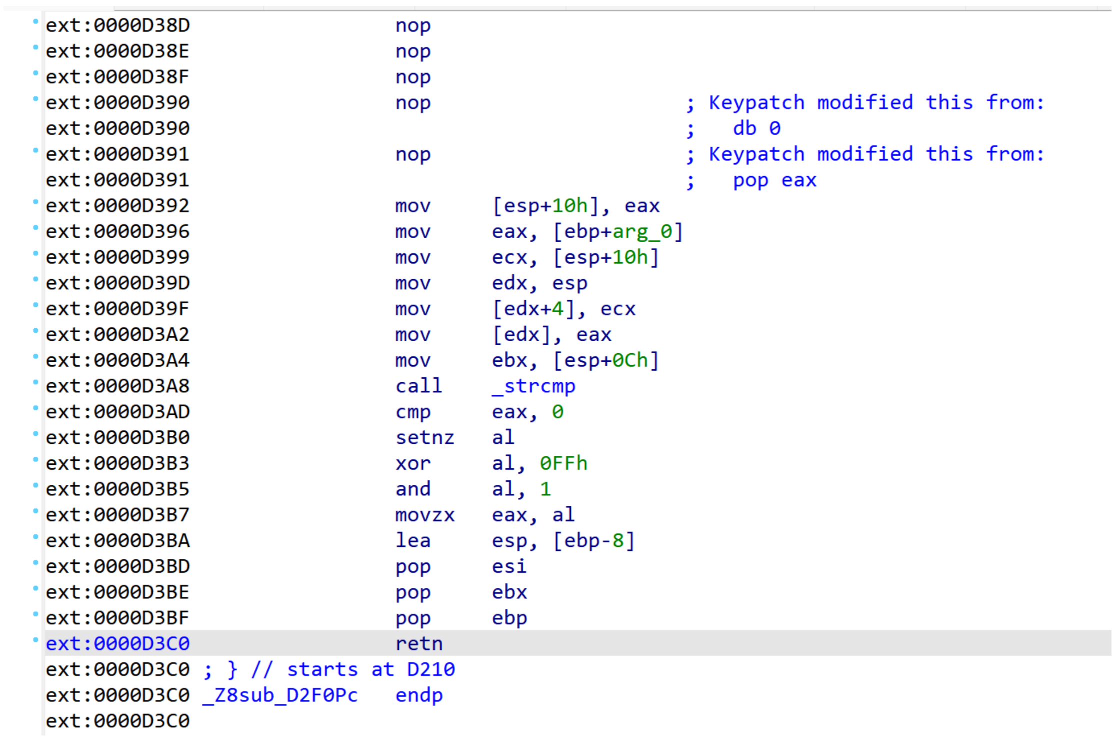


```
0000D383 ; ---------------------------------------------------------------------------
0000D383
0000D383 loc_D383:                              ; CODE XREF: sub_D2F0(char *)+A7↑j
0000D383                 call    sub_D391
0000D388                 nop                    ; Keypatch filled range [0xD388:0xD390] (9 bytes), replace
0000D388                                        ;   db 74h
0000D388                                        ;   db 62h
0000D388                                        ;   db 65h
0000D388                                        ;   db 74h
0000D388                                        ;   db 72h
0000D388                                        ;   db 62h
0000D388                                        ;   db 68h
0000D388                                        ;   db 66h
0000D388                                        ;   db 90h
0000D389                 nop
0000D38A                 nop
0000D38B                 nop
0000D38C                 nop
0000D38D                 nop
0000D38E                 nop
0000D38F                 nop
0000D390                 nop
0000D390 _Z8sub_D2F0Pc   endp ; sp-analysis failed
0000D390
```

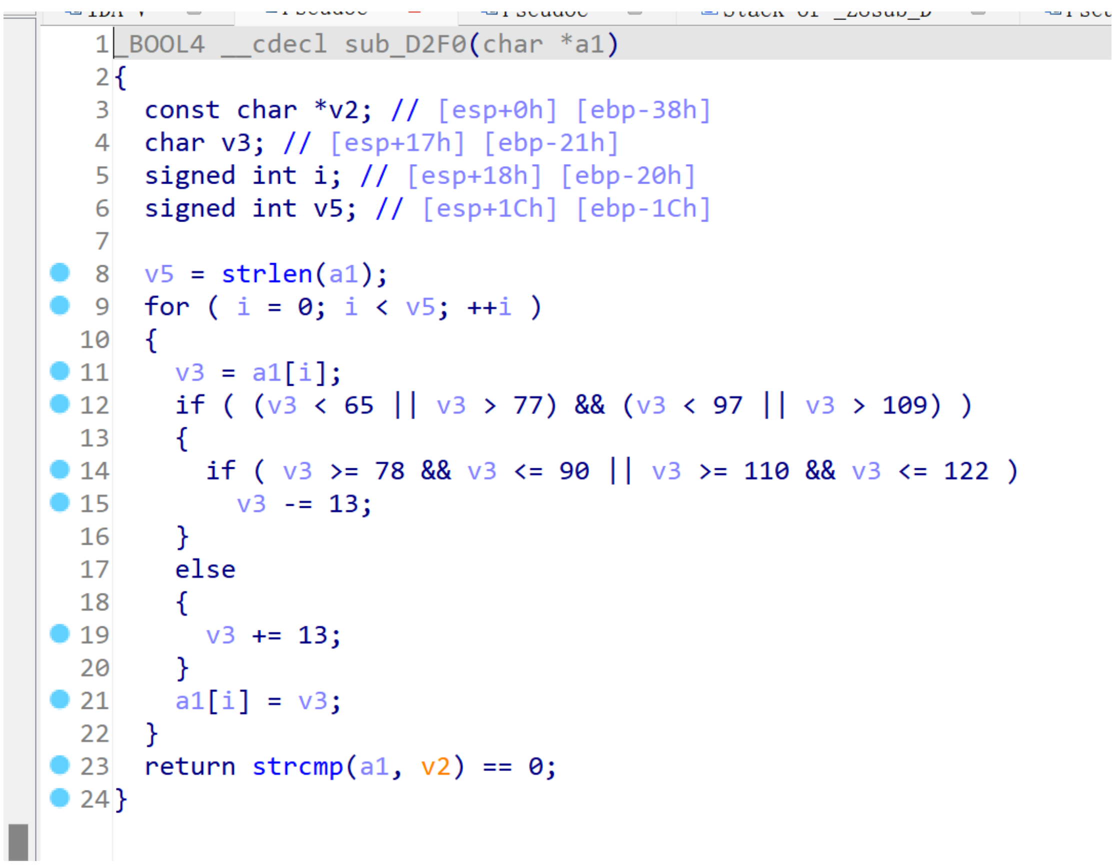
也可直接将 call 和 pop 去除，直接变成 call strcmp。



```
ext:0000D383 loc_D383:                              ; CODE XREF: .text:0000D2B7↑j
ext:0000D383                 nop                    ; Keypatch modified this from:
ext:0000D383                                        ;   call loc_D391
ext:0000D383                                        ; Keypatch padded NOP to next boundary: 4 bytes
ext:0000D384                 nop
ext:0000D385                 nop
ext:0000D386                 nop
ext:0000D387                 nop
ext:0000D388                 nop                    ; Keypatch filled range [0xD388:0xD38F] (8 bytes
ext:0000D388                                        ;   db 74h
ext:0000D388                                        ;   db 62h
ext:0000D388                                        ;   db 65h
ext:0000D388                                        ;   db 74h
ext:0000D388                                        ;   db 72h
ext:0000D388                                        ;   db 62h
ext:0000D388                                        ;   db 68h
ext:0000D388                                        ;   db 66h
ext:0000D389                 nop
ext:0000D38A                 nop
ext:0000D38B                 nop
ext:0000D38C                 nop
ext:0000D38D                 nop
ext:0000D38E                 nop
ext:0000D38F                 nop
ext:0000D390                 nop                    ; Keypatch modified this from:
ext:0000D390                                        ;   db 0
ext:0000D391                 nop                    ; Keypatch modified this from:
ext:0000D391                                        ;   pop eax
```



```
ext:0000D38D                 nop
ext:0000D38E                 nop
ext:0000D38F                 nop
ext:0000D390                 nop                     ; Keypatch modified this from:
ext:0000D390                                         ;   db 0
ext:0000D391                 nop                     ; Keypatch modified this from:
ext:0000D391                                         ;   pop eax
ext:0000D392                 mov     [esp+10h], eax
ext:0000D396                 mov     eax, [ebp+arg_0]
ext:0000D399                 mov     ecx, [esp+10h]
ext:0000D39D                 mov     edx, esp
ext:0000D39F                 mov     [edx+4], ecx
ext:0000D3A2                 mov     [edx], eax
ext:0000D3A4                 mov     ebx, [esp+0Ch]
ext:0000D3A8                 call    _strcmp
ext:0000D3AD                 cmp     eax, 0
ext:0000D3B0                 setnz   al
ext:0000D3B3                 xor     al, 0FFh
ext:0000D3B5                 and     al, 1
ext:0000D3B7                 movzx   eax, al
ext:0000D3BA                 lea     esp, [ebp-8]
ext:0000D3BD                 pop     esi
ext:0000D3BE                 pop     ebx
ext:0000D3BF                 pop     ebp
ext:0000D3C0                 retn
ext:0000D3C0 ; } // starts at D210
ext:0000D3C0 _Z8sub_D2F0Pc   endp
ext:0000D3C0
```

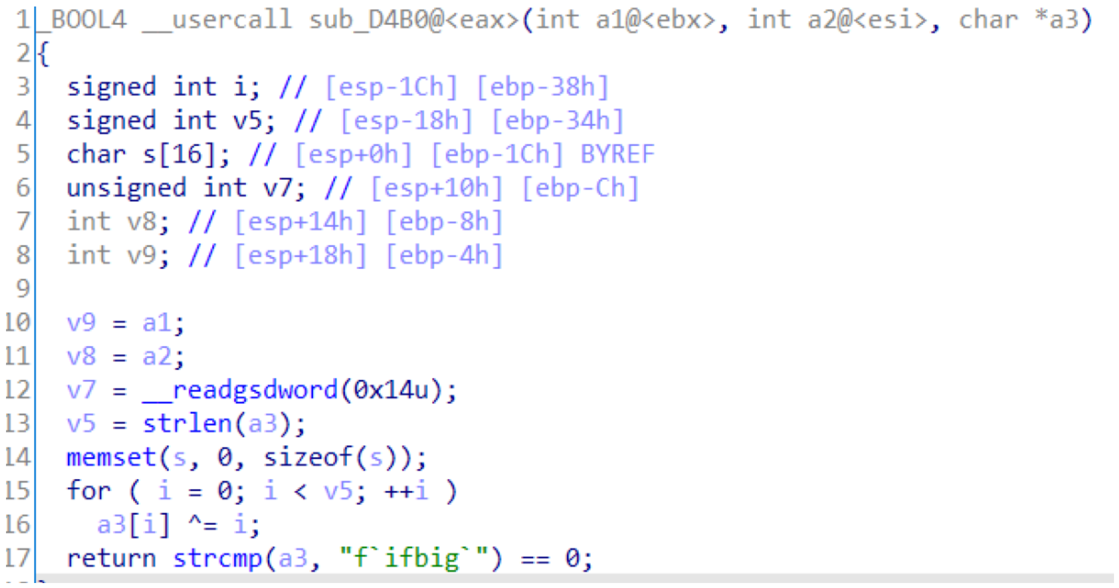


```
_BOOL4 __cdecl sub_D2F0(char *a1)
{
  const char *v2; // [esp+0h] [ebp-38h]
  char v3; // [esp+17h] [ebp-21h]
  signed int i; // [esp+18h] [ebp-20h]
  signed int v5; // [esp+1Ch] [ebp-1Ch]

  v5 = strlen(a1);
  for ( i = 0; i < v5; ++i )
  {
    v3 = a1[i];
    if ( (v3 < 65 || v3 > 77) && (v3 < 97 || v3 > 109) )
    {
      if ( v3 >= 78 && v3 <= 90 || v3 >= 110 && v3 <= 122 )
        v3 -= 13;
    }
    else
    {
      v3 += 13;
    }
    a1[i] = v3;
  }
  return strcmp(a1, v2) == 0;
}
```

可以看出这是一个把 a-m 和 n-z，A-M 和 N-Z 相互转换的程序，所以 tbetrbhf 转换为 gorgeous。

4. 对于 sub_D4B0:

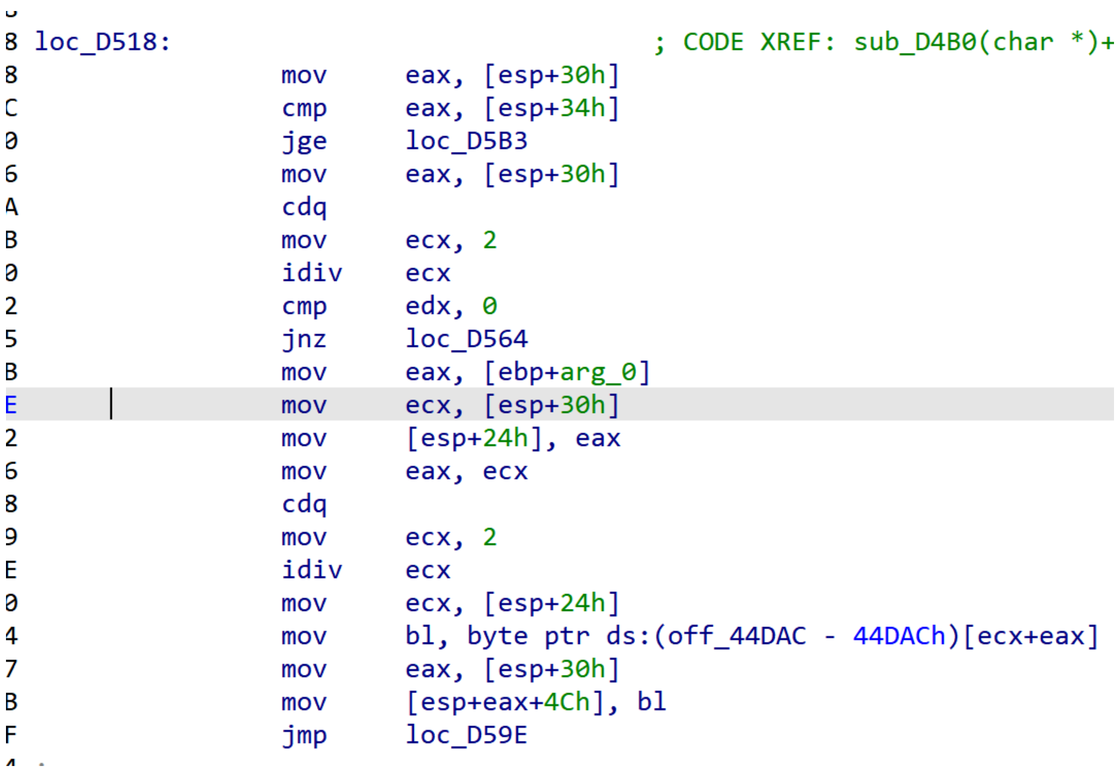
   a) 第一部是先将所有报红的类似上一个函数的处理方法，得到



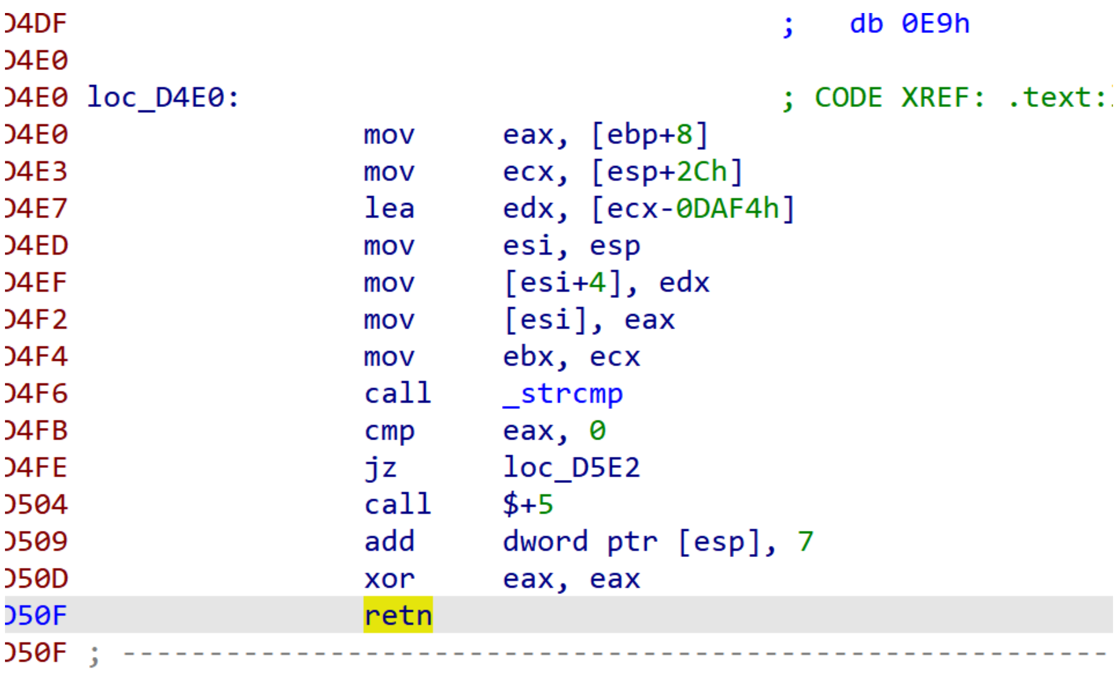
```
_BOOL4 __usercall sub_D4B0@<eax>(int a1@<ebx>, int a2@<esi>, char *a3)
{
  signed int i; // [esp-1Ch] [ebp-38h]
  signed int v5; // [esp-18h] [ebp-34h]
  char s[16]; // [esp+0h] [ebp-1Ch] BYREF
  unsigned int v7; // [esp+10h] [ebp-Ch]
  int v8; // [esp+14h] [ebp-8h]
  int v9; // [esp+18h] [ebp-4h]

  v9 = a1;
  v8 = a2;
  v7 = __readgsdword(0x14u);
  v5 = strlen(a3);
  memset(s, 0, sizeof(s));
  for ( i = 0; i < v5; ++i )
    a3[i] ^= i;
  return strcmp(a3, "f`ifbig`") == 0;
```

   用代码解出结果是“fakeflag”，同前面的输入，结果错误。

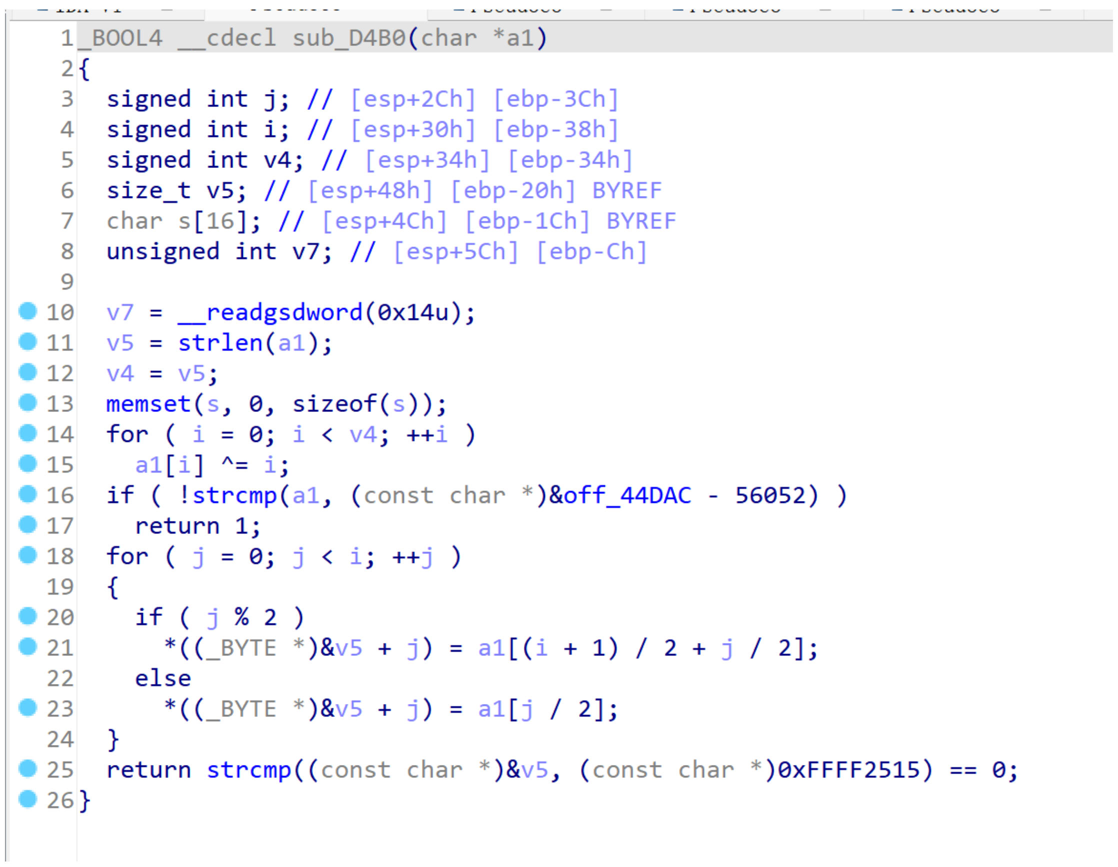
   b) 又重现看汇编代码，发现中间有相当一部分的代码被跳过，主要是 D518 及其上下的函数，根据代码逻辑可见其是一个循环。发现跳过原因是在执行完异或函数后 call 直接 retn，导致无法顺序执行下面的函数，因此将 nop 掉 retn，得到新的函数：



```
8 loc_D518:                              ; CODE XREF: sub_D4B0(char *)+
8                 mov     eax, [esp+30h]
C                 cmp     eax, [esp+34h]
0                 jge     loc_D5B3
6                 mov     eax, [esp+30h]
A                 cdq
B                 mov     ecx, 2
0                 idiv    ecx
2                 cmp     edx, 0
5                 jnz     loc_D564
B                 mov     eax, [ebp+arg_0]
E                 mov     ecx, [esp+30h]
2                 mov     [esp+24h], eax
6                 mov     eax, ecx
8                 cdq
9                 mov     ecx, 2
E                 idiv    ecx
0                 mov     ecx, [esp+24h]
4                 mov     bl, byte ptr ds:(off_44DAC - 44DACh)[ecx+eax]
7                 mov     eax, [esp+30h]
B                 mov     [esp+eax+4Ch], bl
F                 jmp     loc_D59E
```



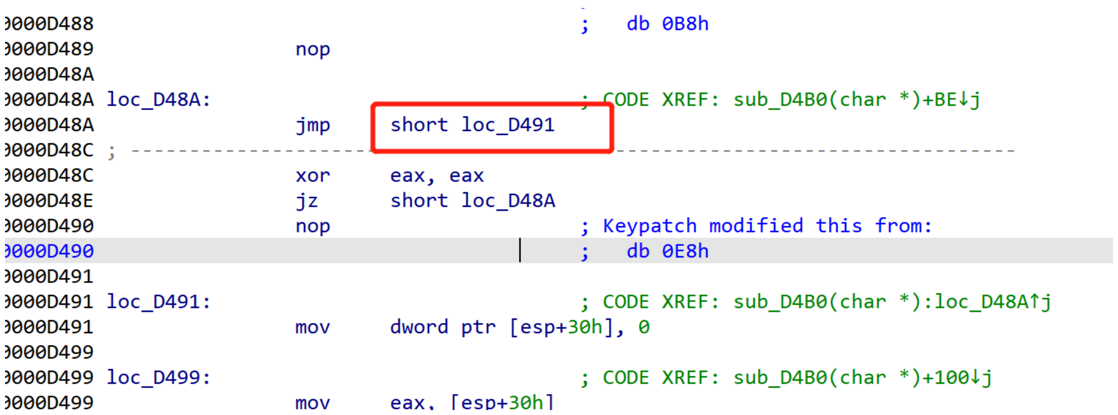
```
4DF                                                     ;   db 0E9h
4E0
4E0 loc_D4E0:                                           ; CODE XREF: .text:
4E0                 mov     eax, [ebp+8]
4E3                 mov     ecx, [esp+2Ch]
4E7                 lea     edx, [ecx-0DAF4h]
4ED                 mov     esi, esp
4EF                 mov     [esi+4], edx
4F2                 mov     [esi], eax
4F4                 mov     ebx, ecx
4F6                 call    _strcmp
4FB                 cmp     eax, 0
4FE                 jz      loc_D5E2
504                 call    $+5
509                 add     dword ptr [esp], 7
50D                 xor     eax, eax
50F                 retn
50F ; ---------------------------------------------------------
```



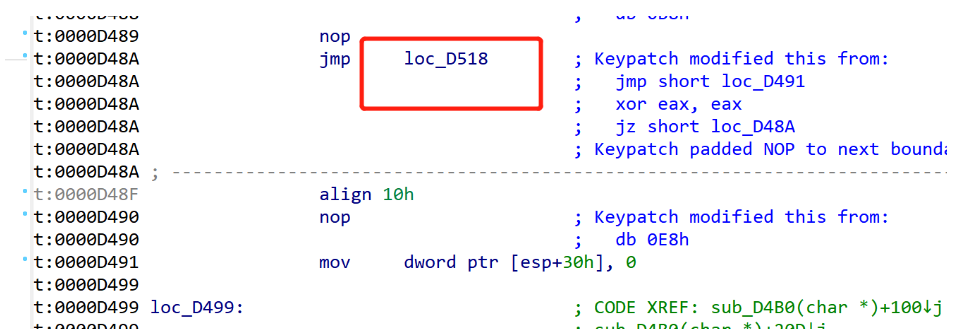
```
_BOOL4 __cdecl sub_D4B0(char *a1)
{
  signed int j; // [esp+2Ch] [ebp-3Ch]
  signed int i; // [esp+30h] [ebp-38h]
  signed int v4; // [esp+34h] [ebp-34h]
  size_t v5; // [esp+48h] [ebp-20h] BYREF
  char s[16]; // [esp+4Ch] [ebp-1Ch] BYREF
  unsigned int v7; // [esp+5Ch] [ebp-Ch]

  v7 = __readgsdword(0x14u);
  v5 = strlen(a1);
  v4 = v5;
  memset(s, 0, sizeof(s));
  for ( i = 0; i < v4; ++i )
    a1[i] ^= i;
  if ( !strcmp(a1, (const char *)&off_44DAC - 56052) )
    return 1;
  for ( j = 0; j < i; ++j )
  {
    if ( j % 2 )
      *((_BYTE *)&v5 + j) = a1[(i + 1) / 2 + j / 2];
    else
      *((_BYTE *)&v5 + j) = a1[j / 2];
  }
  return strcmp((const char *)&v5, (const char *)0xFFFF2515) == 0;
}
```

分析代码发现，新增的循环是隔位取符函数，对结果并没有帮助，依旧是异或的循环决定输入的值。推测在汇编代码中加入了某些花指令，导致两个循环执行顺序相反，因此找到定位的指令，直接修改定位，改变两个循环执行的顺序；

如：

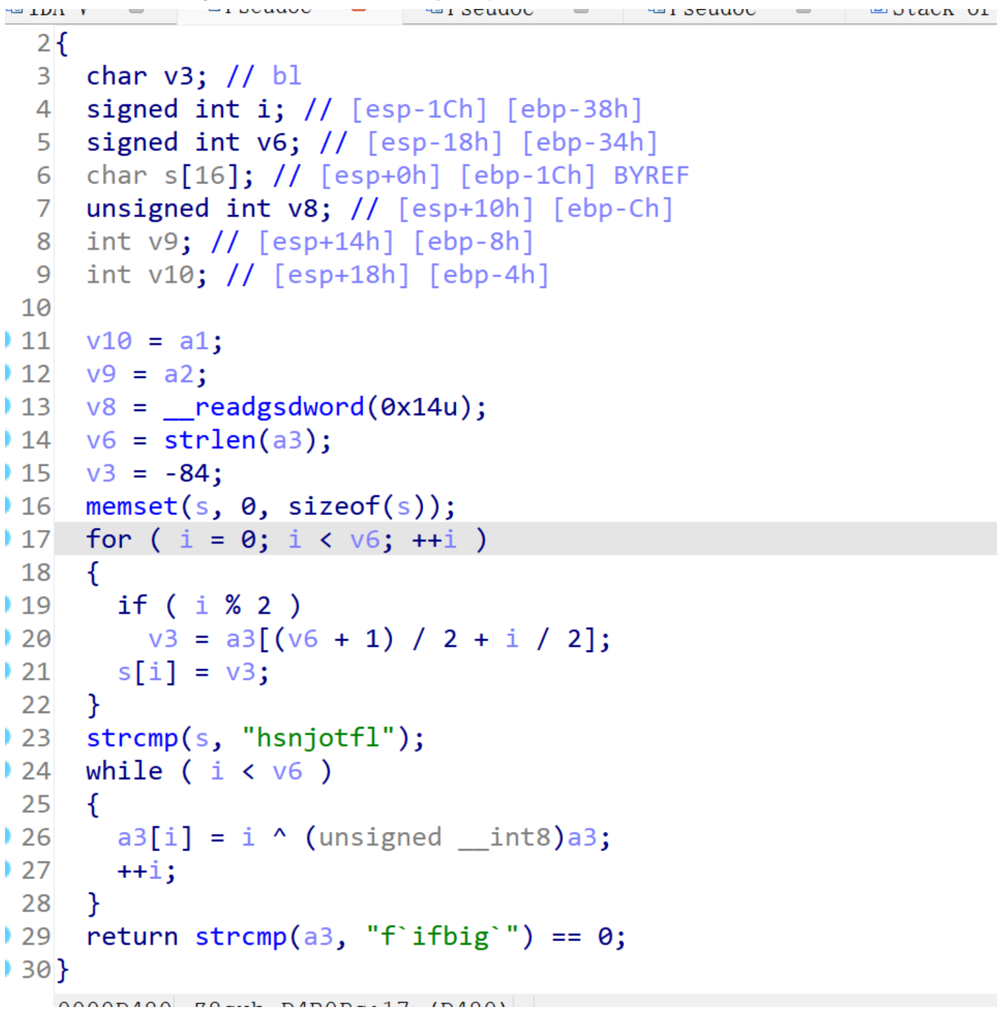


```
000D488                                   ;   db 0B8h
000D489            nop
000D48A
000D48A loc_D48A:                         ; CODE XREF: sub_D4B0(char *)+BE↓j
000D48A            jmp     short loc_D491
000D48C ; ---------------------------------------------------------------------
000D48C            xor     eax, eax
000D48E            jz      short loc_D48A
000D490            nop                    ; Keypatch modified this from:
000D490                                   ;   db 0E8h
000D491
000D491 loc_D491:                         ; CODE XREF: sub_D4B0(char *):loc_D48A↑j
000D491            mov     dword ptr [esp+30h], 0
000D499
000D499 loc_D499:                         ; CODE XREF: sub_D4B0(char *)+100↓j
000D499            mov     eax, [esp+30h]
```



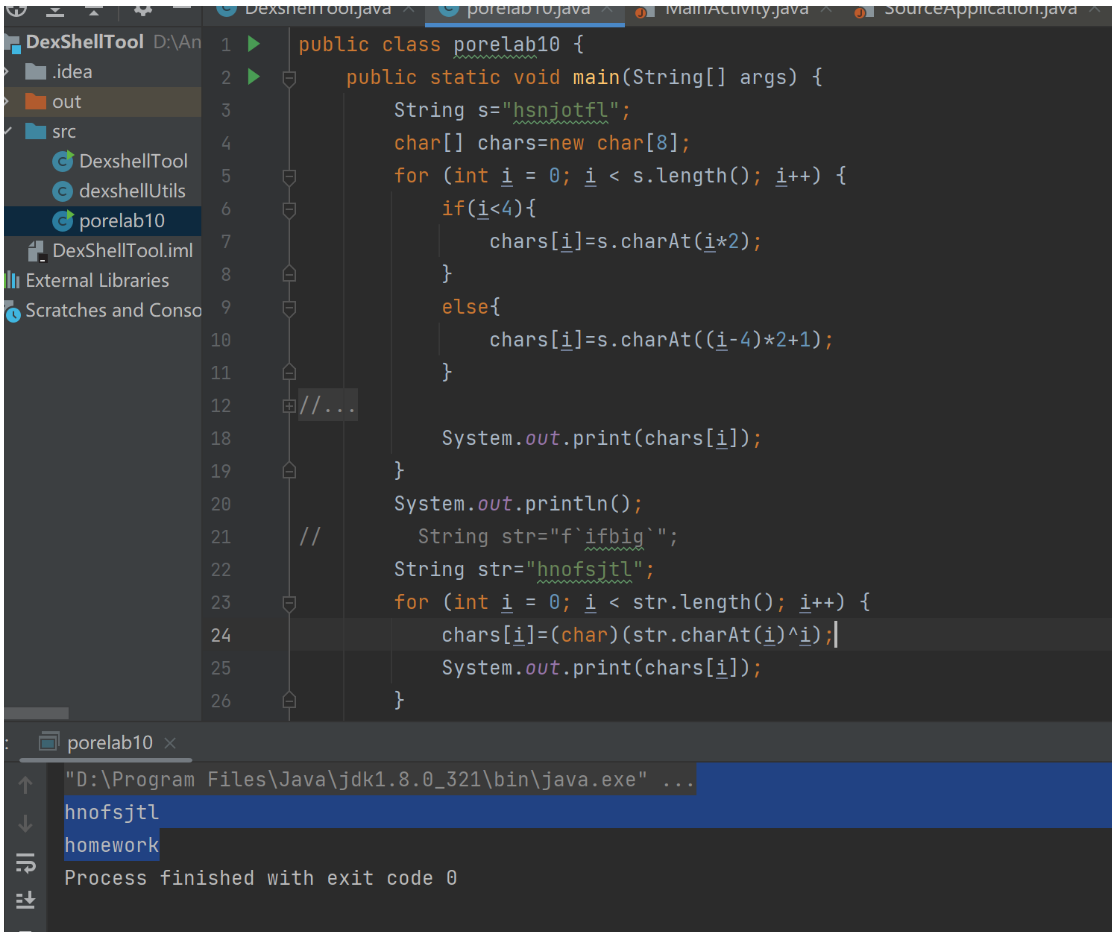
```
t:0000D489            nop
t:0000D48A            jmp     loc_D518      ; Keypatch modified this from:
t:0000D48A                                  ;   jmp short loc_D491
t:0000D48A                                  ;   xor eax, eax
t:0000D48A                                  ;   jz short loc_D48A
t:0000D48A                                  ; Keypatch padded NOP to next bound
t:0000D48A ; ------------------------------------------------------------------
t:0000D48F            align 10h
t:0000D490            nop                   ; Keypatch modified this from:
t:0000D490                                  ;   db 0E8h
t:0000D491            mov     dword ptr [esp+30h], 0
t:0000D499
t:0000D499 loc_D499:                        ; CODE XREF: sub_D4B0(char *)+100↓j
```

后面执行完第二个循环之后也是重定位到异或循环。重新生成函数之后得到：



```
{
  char v3; // bl
  signed int i; // [esp-1Ch] [ebp-38h]
  signed int v6; // [esp-18h] [ebp-34h]
  char s[16]; // [esp+0h] [ebp-1Ch] BYREF
  unsigned int v8; // [esp+10h] [ebp-Ch]
  int v9; // [esp+14h] [ebp-8h]
  int v10; // [esp+18h] [ebp-4h]

  v10 = a1;
  v9 = a2;
  v8 = __readgsdword(0x14u);
  v6 = strlen(a3);
  v3 = -84;
  memset(s, 0, sizeof(s));
  for ( i = 0; i < v6; ++i )
  {
    if ( i % 2 )
      v3 = a3[(v6 + 1) / 2 + i / 2];
    s[i] = v3;
  }
  strcmp(s, "hsnjotfl");
  while ( i < v6 )
  {
    a3[i] = i ^ (unsigned __int8)a3;
    ++i;
  }
  return strcmp(a3, "f`ifbig`") == 0;
}
```

观察函数，发现这样也只是和异或的函数相关，且得到的依旧是 fakeflag；观察整个函数，因为前面的循环得到了原来的 a3 却没有用到，怀疑其实最后的 strcmp 是异或后的 a3 和原来的 a3 进行比较，于是先求出原来的 a3，再代入异或求出最终结果是 homework，这个感觉比较可信，结合前面的 gorgeous 输入 FLAG{gorgeoushomework}，答案正确。



```java
public class porelab10 {
    public static void main(String[] args) {
        String s="hsnjotfl";
        char[] chars=new char[8];
        for (int i = 0; i < s.length(); i++) {
            if(i<4){
                chars[i]=s.charAt(i*2);
            }
            else{
                chars[i]=s.charAt((i-4)*2+1);
            }
//...
            System.out.print(chars[i]);
        }
        System.out.println();
//        String str="f`ifbig`";
        String str="hnofsjtl";
        for (int i = 0; i < str.length(); i++) {
            chars[i]=(char)(str.charAt(i)^i);
            System.out.print(chars[i]);
        }
```

```
"D:\Program Files\Java\jdk1.8.0_321\bin\java.exe" ...
hnofsjtl
homework
Process finished with exit code 0
```