Recursive

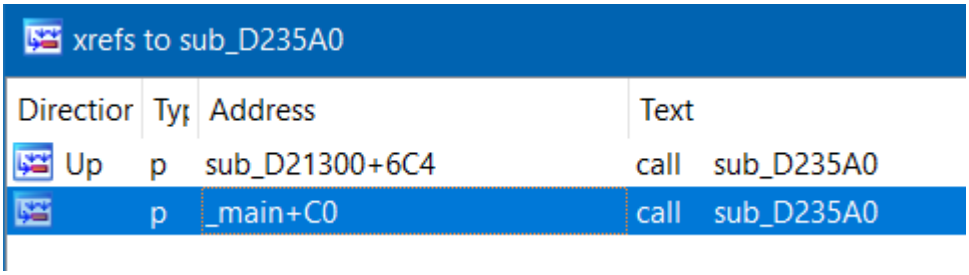
Chạy thử chương trình

```
C:\Users\BILL\Desktop\Release_3\recursive>.\CRACKME.exe
PASSWORD: fsfs
```

Phân tích với IDA, đây là một phần của hàm main

```
LOBYTE(v68) = 2;
sub_D235A0(std::cout, "PASSWORD: ");
v63 = 0;
v64 = 15;
LOBYTE(Src[0]) = 0;
LOBYTE(v68) = 3;
v3 = std::ios::widen(std::cin + *(_DWORD *)(std::cin + 4), 10);
sub_D23B90(Src, std::cin, v3);
v4 = v63;
if ( !v63 )
  exit(1);
v5 = v64;
v6 = Src;
v7 = (void **)Src[0];
if ( v64 >= 0x10 )
  v6 = (void **)Src[0];
if ( *(_BYTE *)v6 == 49 )
{
  v66 = 0;
  v67 = 15;
  LOBYTE(Block[0]) = 0;
  sub_D23190(Block, (void *)"SUCCESS! tell us how this crackme was solved", 0x2Cu);
```

Ta nhận thấy có rất nhiều chuỗi “SUCCESS! tell us how this crackme was solved” trong hàm. Tuy nhiên hàm sub_D23190 không phải là hàm in ra màn hình. Hàm có tác dụng in ra màn hình là hàm sub_D235A0 (vì nó xuất ra chuỗi “PASSWORD: “). Dựa vào đó t tìm được chính xác nơi chúng ta cần đến.



Hàm sub_D235A0 được gọi đúng 2 lần một lần để xuất ra chữ “PASSWORD” một lần còn lại trong hàm sub_D21300 để xuất ra dòng chữ “SUCCESS ...”

```
LOBYTE(v70) = 2;
sub_D235A0(a1, "SUCCESS! tell us how this crackme was solved");
std::ios::clear((char *)a1 + *(_DWORD *)(a1[0] + 4), 0, 0);
sub_D21A80(a1);
```

Đây là nơi chúng ta cần nhảy đến, hàm sub_D235A0 được gọi thẳng từ hàm main.

```
445  v54 = 0;
446  v55 = 15;
447  LOBYTE(v53[0]) = 0;
448  sub_403190(v53, "oh... sorry XD XD XD XD", 0x17u);
449  LOBYTE(v62) = 5;
450  sub_402550(Src);
451  sub_401300();
```

Hàm sub_D235A0 rất dài, tuy nhiên chúng ta suy ngược lên từ câu lệnh gọi sub_D235A0, điều kiện cần đầu tiên là v6 == 6

Đây là đoạn code chúng ta cần quan tâm, vì nó quá dài nên mình không thể chụp hết

```
69  v6 = 0;
70  v70 = 0;
71  v7 = 0;
72  if ( a5 )
73  {
74    while ( v7 )
75    {
76      switch ( v7 )
77      {
78        case 1:
79          sub_D22550(&Src);
80          v10 = Block;
81          v9 = Block[0];
82          if ( v69 >= 0x10 )
83            v10 = (void **)Block[0];
84          if ( *((_BYTE *)v10 + 1) != 116 )
85          {
86            if ( v69 < 0x10 )
87              break;
```

Từ dòng 69 -> 262 hàm sub_D21300

```
226 LABEL_77:
227       if ( ++v7 >= a5 )
228         goto LABEL_78;
229     }
230     sub_D22550(&Src);
231     v8 = Block;
232     v9 = Block[0];
233     if ( v69 >= 0x10 )
234       v8 = (void **)Block[0];
235     if ( *(_BYTE *)v8 != 115 )
236     {
237       if ( v69 < 0x10 )
238         goto LABEL_77;
239       if ( v69 + 1 >= 0x1000 )
240       {
241         v9 = (void *)*((_DWORD *)Block[0] - 1);
242         if ( (unsigned int)(Block[0] - v9 - 4) > 0x1F )
243           goto LABEL_153;
244       }
245       goto LABEL_76;
246     }
247     if ( v69 < 0x10 )
248       goto LABEL_58;
249     if ( v69 + 1 >= 0x1000 )
250     {
251       v9 = (void *)*((_DWORD *)Block[0] - 1);
252       if ( (unsigned int)(Block[0] - v9 - 4) > 0x1F )
253         goto LABEL_153;
254     }
255 LABEL_45:
256     sub_D24015(v9);
257     ++v6;
258     goto LABEL_77;
259   }
260 LABEL_78:
261   if ( v6 == 6 )
262   {
```

Ban đầu v7 = 0, vòng lặp while ở dòng 74 không thực hiện, vì thế từ dòng 230 được thực hiện tiếp, tuy nhiên ở dòng 258 có lệnh nhảy về LABEL_77, lúc này v7 sẽ được tăng lên 1 ở câu lệnh if (++v7 >= a5), từ đó vòng while sẽ thực hiện bắt

đầu từ v7 = 1. Trong đó Src chính là chuỗi chúng ta nhập vào, còn a5 chính là độ dài chuỗi.

Chúng ta cần điều kiện v6 == 6, đồng nghĩa với các điều kiện sau cần:

Các case từ 0 -> 3, chúng ta cần nhảy đến goto LABEL_45 để tăng v6, => giá trị Scr[0:4] =[115, 116,111,112]

```
case 1:
  sub_D22550(&Src);
  v10 = Block;
  v9 = Block[0];
  if ( v69 >= 0x10 )
    v10 = (void **)Block[0];
  if ( *((_BYTE *)v10 + 1) != 116 )
  {
    if ( v69 < 0x10 )
      break;
    if ( v69 + 1 >= 0x1000 )
    {
      v9 = (void *)*((_DWORD *)Block[0] - 1);
      if ( (unsigned int)(Block[0] - v9 - 4) > 0x1F )
        goto LABEL_153;
    }
    goto LABEL_76;
  }
  if ( v69 < 0x10 )
    goto LABEL_58;
  if ( v69 + 1 >= 0x1000 )
  {
    v9 = (void *)*((_DWORD *)Block[0] - 1);
    if ( (unsigned int)(Block[0] - v9 - 4) > 0x1F )
      goto LABEL_153;
  }
  goto LABEL_45;
```

Case 4 và 5, chúng ta cần gọi lệnh ++v6 => giá trị Src[4:6] = [105, 116]

```
  case 4:
    sub_D22550(Block, &Src);
    v13 = Block;
    v14 = Block[0];
    if ( v69 >= 0x10 )
      v13 = (void **)Block[0];
    if ( *((_BYTE *)v13 + 4) == 105 )
    {
      if ( v69 >= 0x10 )
      {
        if ( v69 + 1 >= 0x1000 )
        {
          v14 = (void *)*((_DWORD *)Block[0] - 1);
          if ( (unsigned int)(Block[0] - v14 - 4) > 0x1F )
            goto LABEL_153;
        }
        sub_D24015(v14);
      }
58:
      ++v6;
      break;
    }
```

Toàn bộ 6 điều kiện đều phải đúng thì điều kiện v6 == 6 mới có thể đạt được, từ đó ta tìm ra chuỗi cần nhập.

```
>>> a = [115, 116, 111, 112, 105, 116]
>>> print(''.join(chr(i) for i in a))
stopit
```

Sau điều kiện v6 == 6 thì còn thêm 4 điều kiện khác trước khi nhảy đến câu in ra, tuy nhiên khi mình thử với chuỗi “stopit” thì các điều kiện đều pass vì vậy mình không phân tích các điều kiện còn lại nữa.

```
C:\Users\BILL\Desktop\Release_3\recursive>.\CRACKME.exe
PASSWORD: stopit

C:\Users\BILL\Desktop\Release_3\recursive>cat YJWMaBuRkwIwO
SUCCESS! tell us how this crackme was solved
C:\Users\BILL\Desktop\Release_3\recursive>
```