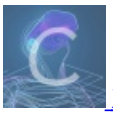
XCTF_BABYRE

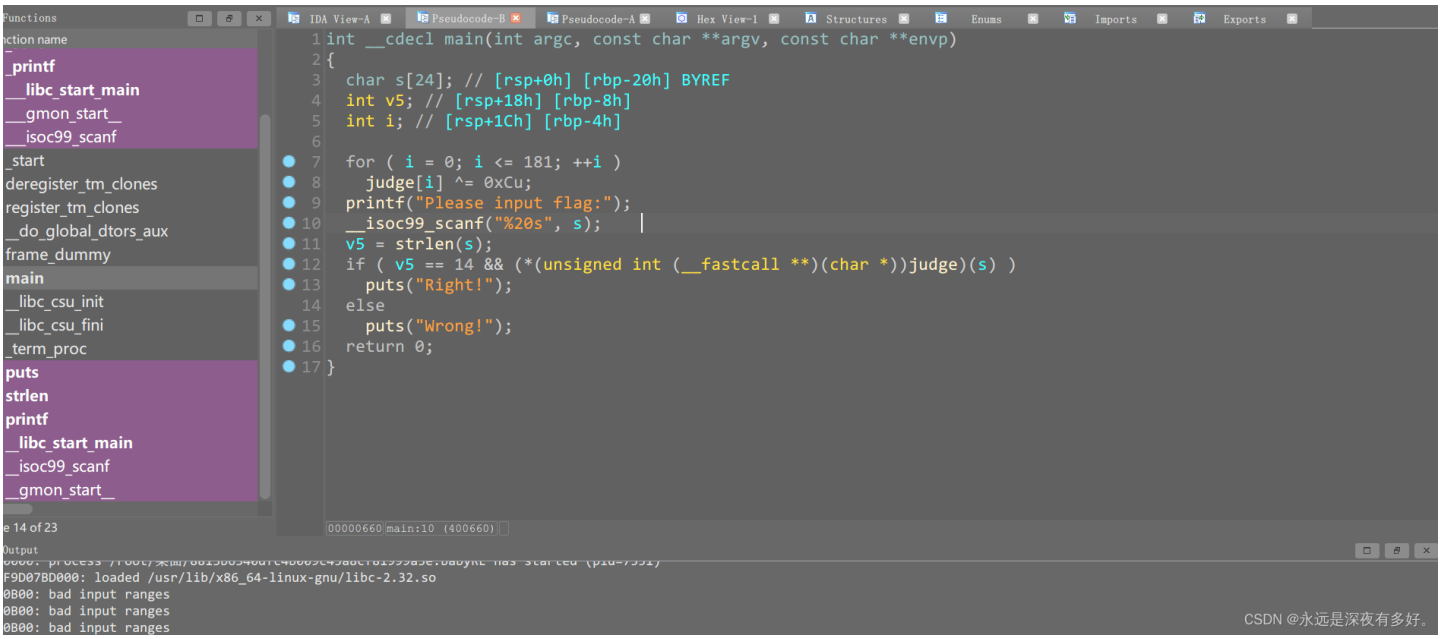
原创

永远是深夜有多好。 于 2022-01-17 21:46:14 发布 178 收藏

分类专栏： XCTF 文章标签： 安全

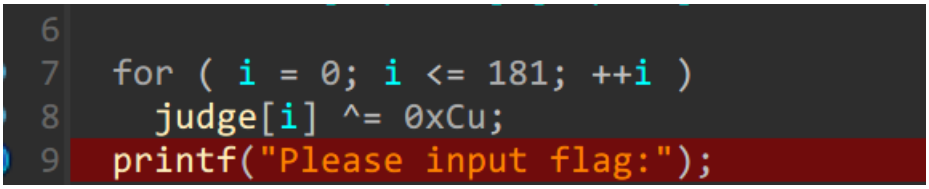
本文链接：https://blog.csdn.net/qq_37370714/article/details/122547694



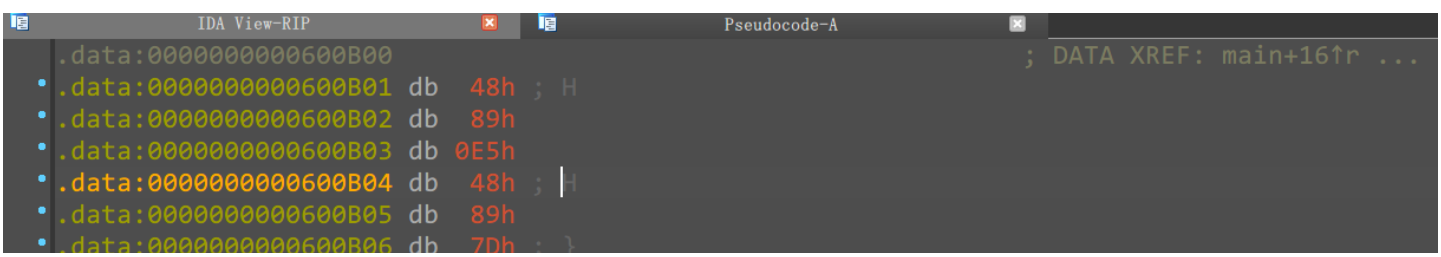


打开发现看不懂也不知道 `judge` 是个啥，只能用我自己会的方法来做试试。

在循环后面下个断点



动态调试后发现 `judge` 值改变了



```
.data:0000000000600B07 db 0D8h
.data:0000000000600B08 db 0C6h
.data:0000000000600B09 db  45h ; E
.data:0000000000600B0A db 0E0h
.data:0000000000600B0B db  66h ; f
.data:0000000000600B0C db 0C6h
.data:0000000000600B0D db  45h ; E
.data:0000000000600B0E db 0E1h
.data:0000000000600B0F db  6Dh ; m
00000B04 0000000000600B04: .data:0000000000600B04 (Synchronized with RIP)
```


现在需要弄清楚这个`judge`是什么
想了一会估计是个函数被怎么处理过就成了数据
重新生成汇编代码看看(按c）

```
.data:0000000000600B00 ; ---------------------------------------------------------------------------
.data:0000000000600B00
.data:0000000000600B00 public judge
.data:0000000000600B00 judge:                                  ; CODE XREF: main+80↑p
.data:0000000000600B00                                         ; DATA XREF: main+16↑r ...
.data:0000000000600B00 push    rbp
.data:0000000000600B01 mov     rbp, rsp
.data:0000000000600B04 mov     [rbp-28h], rdi
.data:0000000000600B08 mov     byte ptr [rbp-20h], 66h ; 'f'
.data:0000000000600B0C mov     byte ptr [rbp-1Fh], 6Dh ; 'm'
.data:0000000000600B10 mov     byte ptr [rbp-1Eh], 63h ; 'c'
.data:0000000000600B14 mov     byte ptr [rbp-1Dh], 64h ; 'd'
.data:0000000000600B18 mov     byte ptr [rbp-1Ch], 7Fh
.data:0000000000600B1C mov     byte ptr [rbp-1Bh], 6Bh ; 'k'
.data:0000000000600B20 mov     byte ptr [rbp-1Ah], 37h ; '7'
.data:0000000000600B24 mov     byte ptr [rbp-19h], 64h ; 'd'
00000B18 0000000000600B18: .data:0000000000600B18 (Synchronized with RIP)
```


哎有东西的样子
鼠标放到public judg`在这里插入代码片`e后面按p重新生成函数

```
a:0000000000600AFE db    0
a:0000000000600AFF db    0
a:0000000000600B00 ; ---------------------------------------------------------------------------
a:0000000000600B00
a:0000000000600B00 public judge
a:0000000000600B00 judge:
a:0000000000600B00
```

重新生成c代码发现这应该是

```
{
  char v2[5]; // [rsp+8h] [rbp-20h] BYREF
  char v3[9]; // [rsp+Dh] [rbp-1Bh] BYREF
  int i; // [rsp+24h] [rbp-4h]

  qmemcpy(v2, "fmcd", 4);
  v2[4] = 127;
  qmemcpy(v3, "k7d;V`;np", sizeof(v3));
  for ( i = 0; i <= 13; ++i )
    *(_BYTE *)(i + a1) ^= i;
```

```
for ( i = 0; i <= 13; ++i )
{
  if ( *(_BYTE *)(i + a1) != v2[i] )
    return 0LL;
}
return 1LL;
```



```
#include <stdio.h>
#include <string.h>
#pragma warning(disable:4996)
int main()
{
char v2[15]= { 0 };
char flag[15] = { 0 };
strcat(v2,"fmcd");
v2[4] = 127;
strcat(v2, "k7d;V`;np");
//printf("%d\n", sizeof(v2)); v2大小为15
for (int i = 0; i <= 13; ++i)
    flag[i]=v2[i]^i;
printf("%s", flag);
}
```

Microsoft Visual Studio 调试控制台

```
flag{n1c3_j0b}
```


后面百度和看别人的wp发现这道题相关知识点是SMC(Self-Modifying-Code)代码自修改，是一种反调试手法。
新知识get