# 关于2022虎符pwn mva解决jmp rax无法反汇编的这么个事情

原创

yongbaoii 于 2022-03-23 20:47:50 发布 449 收藏 3

分类专栏： CTF 文章标签： 网络安全



本文链接：https://blog.csdn.net/yongbaoii/article/details/123675415





事情的起源是因为

2022 虎符 pwn mva 题解

那么有个问题来了

程序整个是个vm

但是在跳转的时候用了jmp rax导致它的十六个功能无法反汇编出来

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  unsigned __int16 v4; // [rsp+20h] [rbp-240h]

  sub_1277(a1, a2, a3);
  puts("[+] Welcome to MVA, input your code now :");
  fread(&unk_4040, 0x100uLL, 1uLL, stdin);
  puts("[+] MVA is starting ...");
  while ( 1 )
  {
    v4 = ((unsigned int (*)(void))((char *)&sub_11E8 + 1))() >> 24;
    if ( v4 > 0xFu )
      break;
    if ( v4 <= 0xFu )
      __asm { jmp     rax }
  }
  puts("[+] MVA is shutting down ...");
  return 0LL;
}
```



当时比赛的时候是去硬读汇编解决的。

但是思索着有没有解决办法

最后经过无敌的cherest_san师傅帮忙解决了这个问题。特此记录。

~~他让不会的人都去问他~~

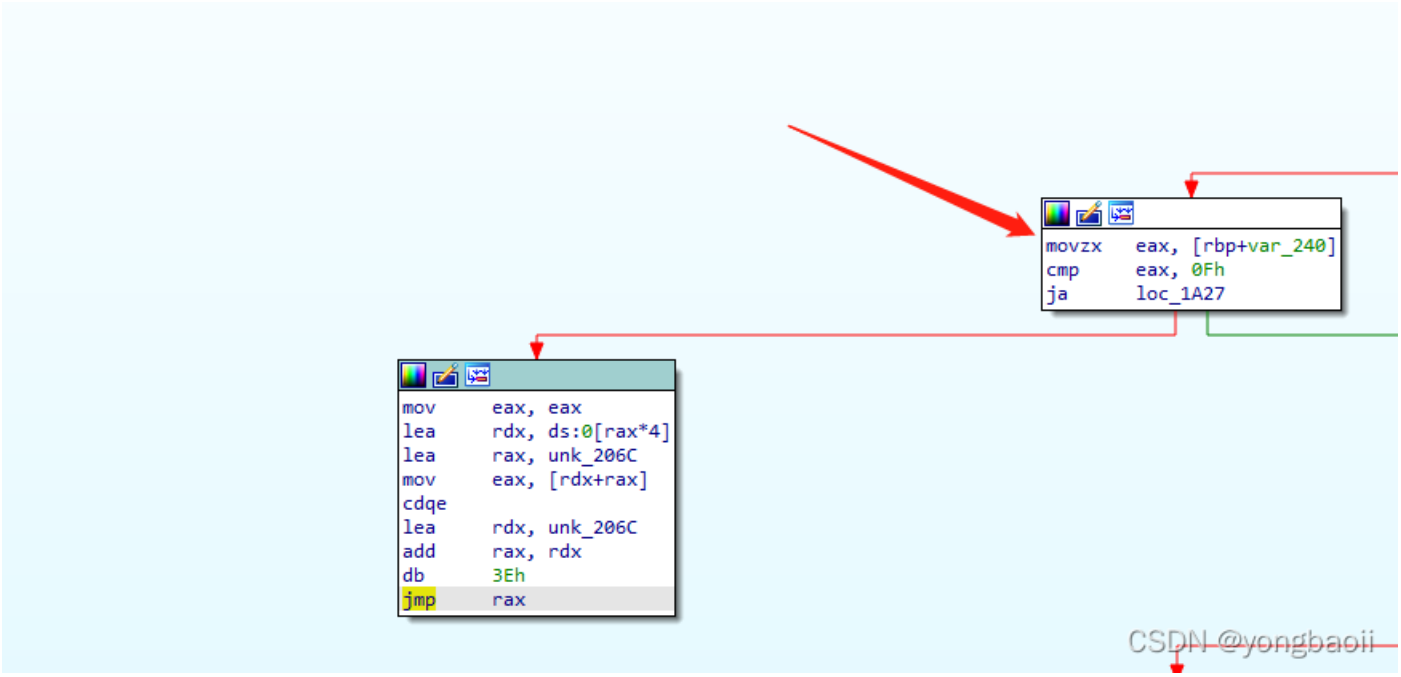
他说事了拂衣去，深藏功与名，但是高尚的我还是决定提他一嘴。

首先我们要清楚这里反汇编不出来主要是因为它没有跳到一个确定地址的地方。

那我们能不能通过适当的patch程序，来改变这里的逻辑。让他能直接跳到确定地址地方。

我们patch的程序不一定能跑，也不需要能跑，只要IDA能反汇编出来就成。

整体逻辑呢

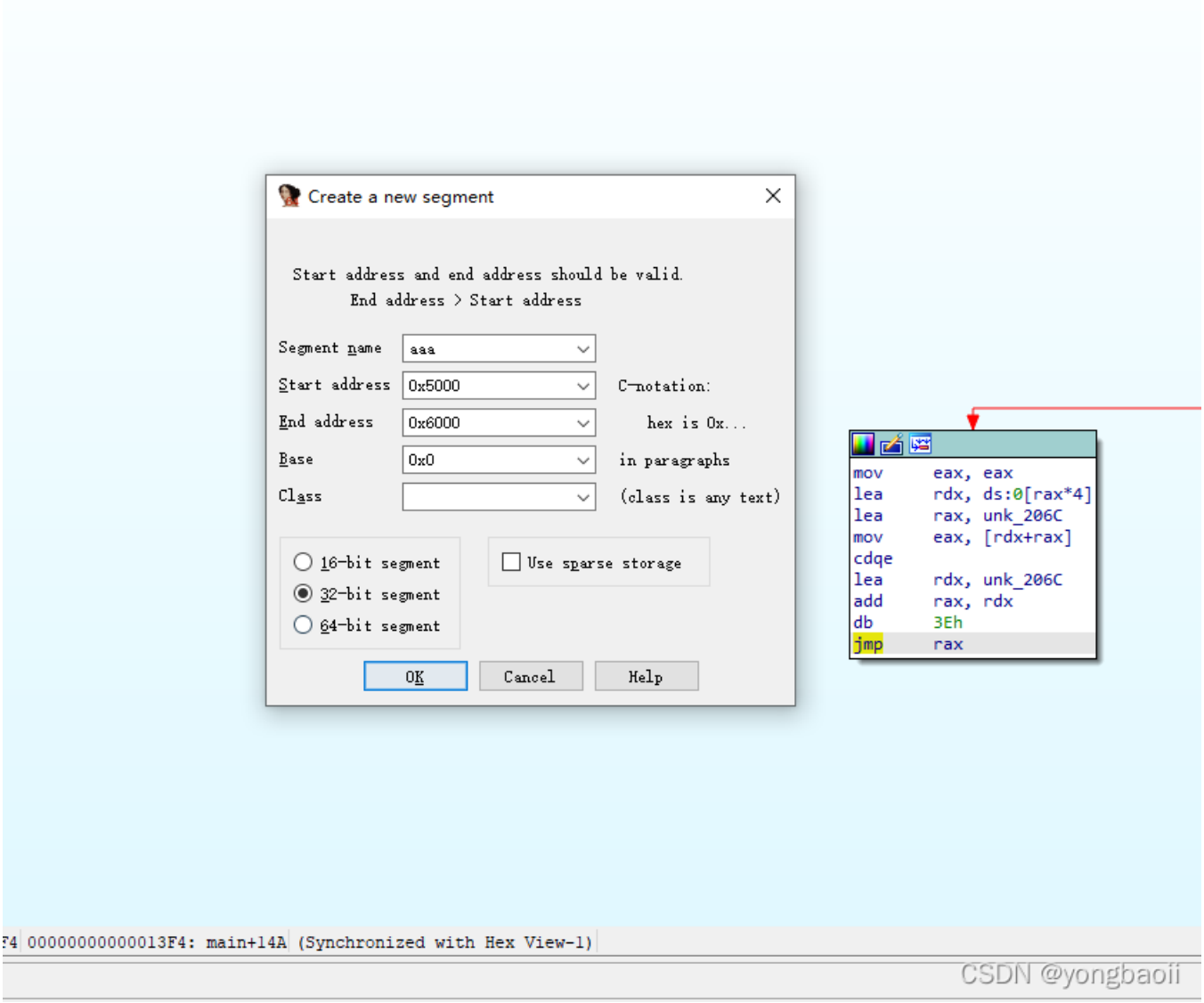



是让这里的返回值rax在跟0xf比较完之后立马跳到我们新开的段

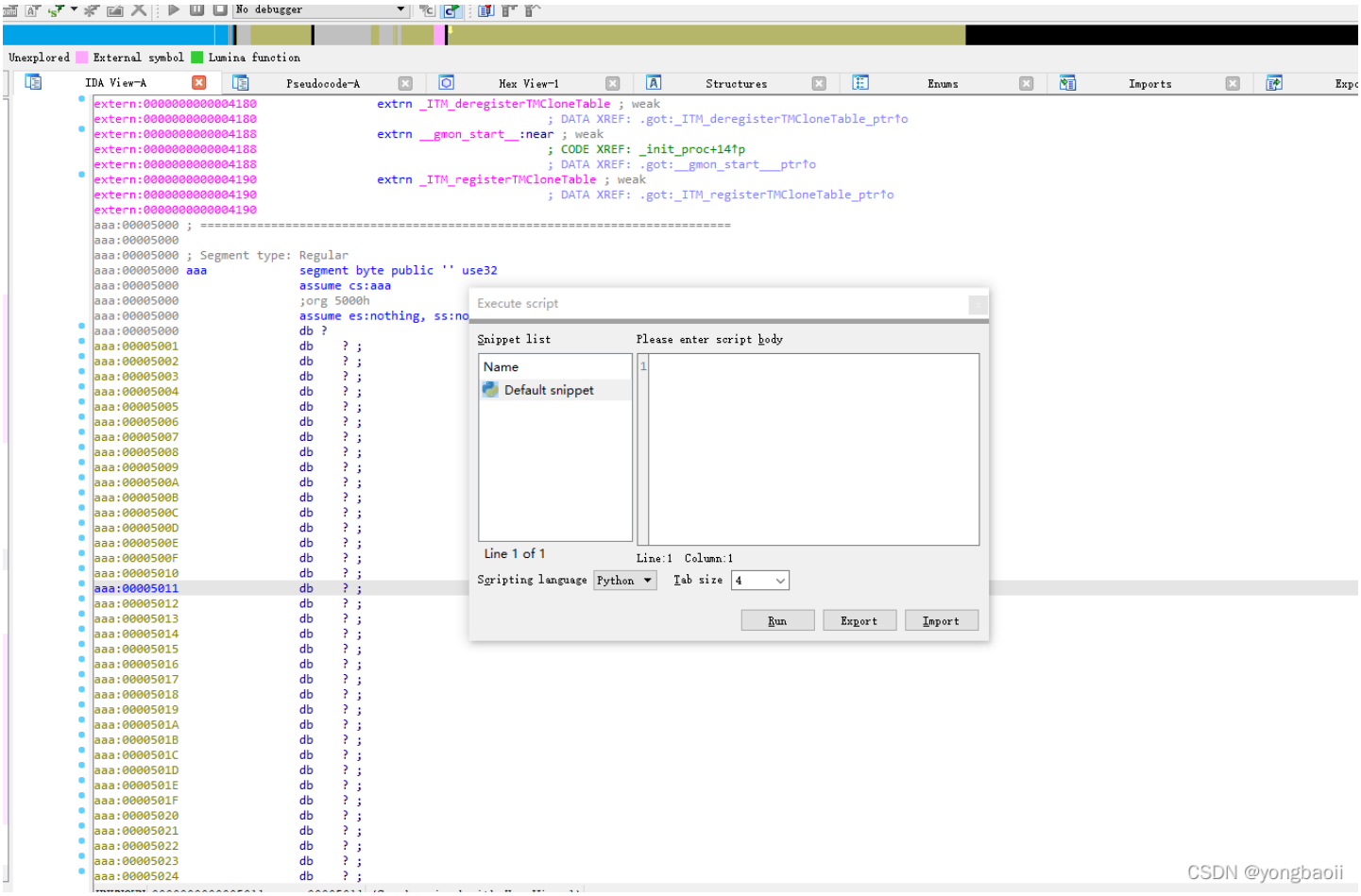
然后用汇编不停的cmp if else

第一步先开段

edit -> segment -> creat segment




这样直接开个段。




然后用IDApython
跑这个脚本。

```
to=[0x13f7,0x1405,0x143e,0x14cc,0x155c
,0x15ea,0x1678,0x16f2,0x1780
,0x17a1,0x1813,0x1876,0x18a5
,0x1920,0x19ac,0x1a00]
base=0x100000000
addr = 0x5000
for i in range(15):
    PatchWord(addr,0xf883)
    PatchByte(addr+2,i)
    PatchWord(addr+3,0x0575)
    PatchByte(addr+5,0xe9)
    jmp = base + to[i] - (addr + 10)
    PatchDword(addr+6,jmp)
    addr += 10
PatchByte(addr,0xe9)
jmp = base+to[15]-(addr+5)
PatchDword(addr+1,jmp)
```

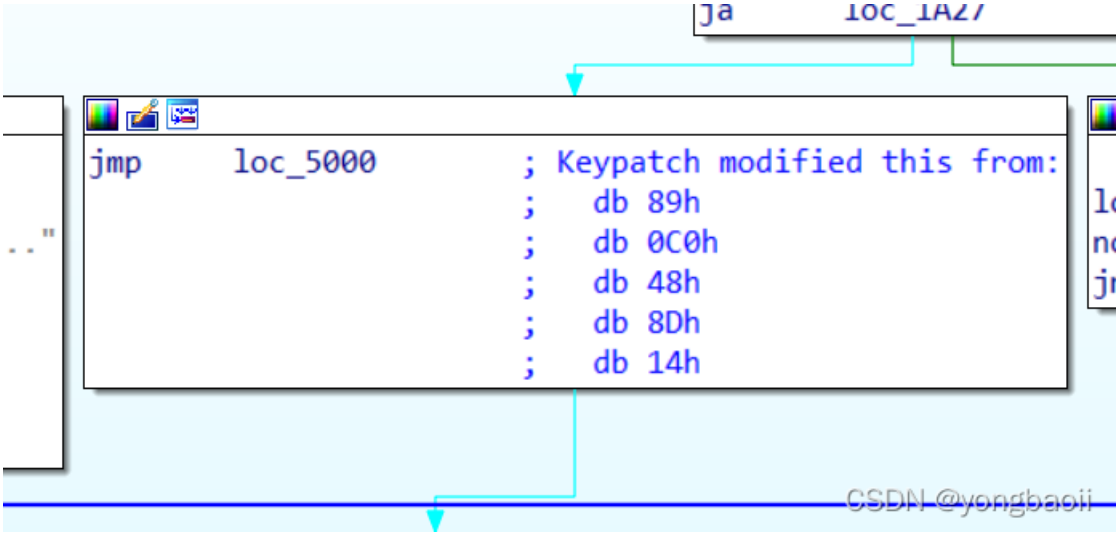
脚本意思很简单
就是通过一个循环
将0x5000处patch成这样子
每个循环里面是一个cmp、一个jnz、一个cmp。

```
aaa:0000000000005000 loc_5000:                           ; CODE XREF: main+12A↑j
aaa:0000000000005000                 cmp     eax, 0
aaa:0000000000005003                 jnz     short loc_500A
aaa:0000000000005005                 jmp     loc_13F7
aaa:000000000000500A ; ---------------------------------------------------------------------------
aaa:000000000000500A
aaa:000000000000500A loc_500A:                           ; CODE XREF: aaa:0000000000005003↑j
aaa:000000000000500A                 cmp     eax, 1
aaa:000000000000500D                 jnz     short loc_5014
aaa:000000000000500F                 jmp     loc_1405
aaa:0000000000005014 ; ---------------------------------------------------------------------------
aaa:0000000000005014
aaa:0000000000005014 loc_5014:                           ; CODE XREF: aaa:000000000000500D↑j
aaa:0000000000005014                 cmp     eax, 2
aaa:0000000000005017                 jnz     short loc_501E
aaa:0000000000005019                 jmp     loc_143E
aaa:000000000000501E ; ---------------------------------------------------------------------------
aaa:000000000000501E
aaa:000000000000501E loc_501E:                           ; CODE XREF: aaa:0000000000005017↑j
aaa:000000000000501E                 cmp     eax, 3
aaa:0000000000005021                 jnz     short loc_5028
aaa:0000000000005023                 jmp     loc_14CC
aaa:0000000000005028 ; ---------------------------------------------------------------------------
aaa:0000000000005028
aaa:0000000000005028 loc_5028:                           ; CODE XREF: aaa:0000000000005021↑j
aaa:0000000000005028                 cmp     eax, 4
aaa:000000000000502B                 jnz     short loc_5032
aaa:000000000000502D                 jmp     loc_155C
aaa:0000000000005032 ; ---------------------------------------------------------------------------
aaa:0000000000005032
aaa:0000000000005032 loc_5032:                           ; CODE XREF: aaa:000000000000502B↑j
aaa:0000000000005032                 cmp     eax, 5
aaa:0000000000005035                 jnz     short loc_503C
aaa:0000000000005037                 jmp     loc_15EA
aaa:000000000000503C ; ---------------------------------------------------------------------------
```



最后把这里用keypatch改掉。

跳过去。刚刚那个段地址是0x5000。




最后的效果是这样的

```
unsigned __int16 v6; // [rsp+20h] [rbp-240h]
 __int8 code[4]; // [rsp+24h] [rbp-23Ch]
 int v8; // [rsp+28h] [rbp-238h]
 __int64 RRSP; // [rsp+30h] [rbp-230h]
 _WORD reg[6]; // [rsp+44h] [rbp-21Ch] BYREF
 _WORD v11[260]; // [rsp+50h] [rbp-210h]
 unsigned __int64 v12; // [rsp+258h] [rbp-8h]

 v12 = __readfsqword(0x28u);
 sub_1277();
 v4 = 0;
 RRSP = 0LL;
```

```
memset(reg, 0, sizeof(reg));
v5 = 1;
puts("[+] Welcome to MVA, input your code now :");
fread(&unk_4040, 0x100uLL, 1uLL, stdin);
puts("[+] MVA is starting ...");
while ( v5 )
{
  *(_DWORD *)code = encode();
  v6 = HIBYTE(*(_DWORD *)code);
  if ( v6 > 0xFu )
    break;
  if ( v6 <= 0xFu )
  {
    if ( v6 )
    {
      switch ( v6 )
      {
        case 1u:
          if ( code[2] > 5 || code[2] < 0 )
            exit(0);
          reg[code[2]] = *(_WORD *)code;
          break;
        case 2u:
          if ( code[2] > 5 || code[2] < 0 )
            exit(0);
          if ( code[1] > 5 || code[1] < 0 )
            exit(0);
          if ( code[0] > 5 || code[0] < 0 )
            exit(0);
          reg[code[2]] = reg[code[1]] + reg[code[0]];
          break;
        case 3u:
          if ( code[2] > 5 || code[2] < 0 )
            exit(0);
          if ( code[1] > 5 || code[1] < 0 )
            exit(0);
          if ( code[0] > 5 || code[0] < 0 )
            exit(0);
          reg[code[2]] = reg[code[1]] - reg[code[0]];
          break;
        case 4u:
          if ( code[2] > 5 || code[2] < 0 )
            exit(0);
          if ( code[1] > 5 || code[1] < 0 )
            exit(0);
          if ( code[0] > 5 || code[0] < 0 )
            exit(0);
          reg[code[2]] = reg[code[1]] & reg[code[0]];
          break;
        case 5u:
          if ( code[2] > 5 || code[2] < 0 )
            exit(0);
          if ( code[1] > 5 || code[1] < 0 )
            exit(0);
          if ( code[0] > 5 || code[0] < 0 )
            exit(0);
          reg[code[2]] = reg[code[1]] | reg[code[0]];
          break;
        case 6u:
          if ( code[2] > 5 || code[2] < 0 )
```

```
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( code[1] > 5 || code[1] < 0 )
      exit(0);
    reg[code[2]] = (int)(unsigned __int16)reg[code[2]] >> reg[code[1]];
    break;
  case 7u:
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( code[1] > 5 || code[1] < 0 )
      exit(0);
    if ( code[0] > 5 || code[0] < 0 )
      exit(0);
    reg[code[2]] = reg[code[1]] ^ reg[code[0]];
    break;
  case 8u:
    RRIP = encode();
    break;
  case 9u:
    if ( RRSP > 0x100 )
      exit(0);
    if ( code[2] )
      v11[RRSP] = *(_WORD *)code;
    else
      v11[RRSP] = reg[0];
    ++RRSP;
    break;
  case 0xAu:
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( !RRSP )
      exit(0);
    reg[code[2]] = v11[--RRSP];
    break;
  case 0xBu:
    v8 = encode();
    if ( v4 == 1 )
      RRIP = v8;
    break;
  case 0xCu:
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( code[1] > 5 || code[1] < 0 )
      exit(0);
    v4 = reg[code[2]] == reg[code[1]];
    break;
  case 0xDu:
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( code[0] > 5 || code[0] < 0 )
      exit(0);
    reg[code[2]] = reg[code[1]] * reg[code[0]];
    break;
  case 0xEu:
    if ( code[2] > 5 || code[2] < 0 )
      exit(0);
    if ( code[1] > 5 )
      exit(0);
    reg[code[1]] = reg[code[2]];
    break;
```

```
          default:
            printf("%d\n", (unsigned __int16)v11[RRSP]);
            break;
        }
      }
      else
      {
        v5 = 0;
      }
    }
  }
  puts("[+] MVA is shutting down ...");
  return 0LL;
}
```

伪代码整体逻辑清晰
但是其实反而不大好找整数溢出的漏洞
汇编要读的快读汇编也行。