简陋的VM

发表于 2021-03-05 分类于 Challenge ， 2020 ， SWPU CTF 2020 ， Reverse
Challenge | 2020 | SWPU CTF 2020 | Reverse | 简陋的VM

点击此处获得更好的阅读体验

WriteUp来源

官方WP

https://www.cnblogs.com/mio-yy/p/14141440.html

题目考点

- *VM逆向*

解题思路

官方WP

这道题其实就是一个简单的使用c语言模拟执行的vm，本身并没有做什么处理，只需要找到分发器的位置，再将每一条handle解析出来就能还原汇编。

首先是分发器，利用的是push/ret的方式调用的，获取需要call的函数地址，将参数与返回地址依次入栈，最后压入跳转地址ret就可以了，所以需要动态调试一下。

```
1  add_call = call_list[vm_call];
2  __asm
3  {
4   push i;
5   sub esp, 0x4;
6   mov eax, v_ret;
7   mov dword ptr ss : [esp] , eax;
8   push add_call;
9   ret;
10 v_ret:
11  mov i, eax;
12  }
```

接下来就是解析每个handle，原本的汇编指令应该如下所示：

```
signed opcode[219] = {
  op_mov, 6, op_ebp, op_esp,
  op_sub, op_esp, 54,
  op_push, op_esi,               //2 push esi
  op_push, op_edi,               //2 push edi
  op_mov, 4, op_ecx, 9,            //4 mov ecx,0x9
  op_mov, 0x10, op_esi,            //3 mov esi,Project2.00C620F8
  op_lea, op_edi, -0x50,           //3 lea edi,dword ptr ss:[ebp-0x50]
  op_rep_movs,                  //1 rep movs dword ptr es:[edi],dword ptr ds:[esi]
  op_mov, 5, -0x54, 0,           //4 mov dword ptr ss:[ebp-0x54],0x0
  op_jmp,  12,                  //2 jmp short Project2.00C61033
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_add, op_eax, 0x1,           //3 add eax,0x1
  op_mov, 5, -0x54, op_eax,         //4 mov dword ptr ss:[ebp-0x54],eax
  op_cmp, -0x54, 0x24,           //3 cmp dword ptr ss:[ebp-0x54],0x24
  op_jge, 89,                  //2 jge Project2.00C610D6
  op_mov, 3, op_ecx, -0x54,         //4 mov ecx,dword ptr ss:[ebp-0x54]
  op_and, op_ecx, 0x80000001,         //3 and ecx,0x80000001
  op_jns, 8,                  //2 jns short Project2.00C6104D
  op_dec, op_ecx,               //2 dec ecx
  op_or, op_ecx, -0x2,           //3 or ecx,-0x2
  op_inc, op_ecx,               //2 inc ecx
  op_test, op_ecx, op_ecx,          //3 test ecx,ecx
  op_je, 35,                  //2 je short Project2.00C6106F
  op_mov, 3, op_edx, -0x54,         //4 mov edx,dword ptr ss:[ebp-0x54]
  op_mov, 0x10B, op_eax, op_edx, -0x50,    //5 movzx eax,byte ptr ss:[ebp+edx-0x50]
  op_add, op_eax, 0x5,           //3 add eax,0x5
  op_mov, 4, op_ecx, 0x68,           //4 mov ecx,0x68
  op_sub, 3, op_ecx, -0x54,         //4 sub ecx,dword ptr ss:[ebp-0x54]
  op_xor, op_eax, op_ecx,           //3 xor eax,ecx
  op_mov, 3, op_edx, -0x54,         //4 mov edx,dword ptr ss:[ebp-0x54]
  op_mov, 0x10D, op_edx, -0x2c, op_eax,    //5 mov byte ptr ss:[ebp+edx-0x2C],al
  op_jmp,  32,                  //2 jmp short Project2.00C61089
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_mov, 0x10B, op_ecx, op_eax, -0x50,    //5 movzx ecx,byte ptr ss:[ebp+eax-0x50]
  op_sub, op_ecx, 0x3,           //3 sub ecx,0x3
  op_mov, 3, op_edx, -0x54,         //4 mov edx,dword ptr ss:[ebp-0x54]
  op_add, op_edx, 0x67,           //3 add edx,0x67
  op_xor, op_ecx, op_edx,           //3 xor ecx,edx
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_mov, 0x10D, op_eax, -0x2c, op_ecx,     //5 mov byte ptr ss:[ebp+eax-0x2C],cl
  op_jmp,  -103,                //2 jmp
  op_mov, 5, -0x54, 0x0,           //4 mov dword ptr ss:[ebp-0x54],0x0
  op_jmp,  12,                  //2 jmp short Project2.00C6109B
  op_mov, 3, op_ecx, -0x54,         //4 mov ecx,dword ptr ss:[ebp-0x54]
  op_add, op_ecx, 0x1,           //3 add ecx,0x1
  op_mov, 5, -0x54, op_ecx,         //4 mov dword ptr ss:[ebp-0x54],ecx
  op_cmp, -0x54, 0x12,           //3 cmp dword ptr ss:[ebp-0x54],0x12
  op_jge,  59,                  //2 jge short Project2.00C610D1
  op_mov, 3, op_edx, -0x54,         //4 mov edx,dword ptr ss:[ebp-0x54]
  op_add, op_edx, 0x32,           //3 add edx,0x32
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_mov, 0x10B, op_ecx, op_eax, -0x2c,     //5 movzx ecx,byte ptr ss:[ebp+eax-0x2C]
  op_xor, op_ecx, op_edx,           //3 xor ecx,edx
  op_mov, 3, op_edx, -0x54,         //4 mov edx,dword ptr ss:[ebp-0x54]
  op_mov, 0x10D, op_edx, -0x2c, op_ecx,    //5 mov byte ptr ss:[ebp+edx-0x2C],cl
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_add, op_eax, 0x23,           //3 add eax,0x23
  op_mov, 3, op_ecx, -0x54,         //4 mov ecx,dword ptr ss:[ebp-0x54]
  op_mov, 0x10B, op_edx, op_ecx, -0x1a,    //5 movzx edx,byte ptr ss:[ebp+ecx-0x1A]
  op_xor, op_edx, op_eax,           //3 xor edx,eax
  op_mov, 3, op_eax, -0x54,         //4 mov eax,dword ptr ss:[ebp-0x54]
  op_mov, 0x10D, op_eax, -0x1a,  op_edx,    //5 mov byte ptr ss:[ebp+eax-0x1A],dl
  op_jmp,  -73,                 //2 jmp short Project2.00C61092
  op_xor, op_eax, op_eax,           //3 xor eax,eax
  op_pop, op_edi,               //2 pop edi
  op_pop, op_esi,               //2 pop esi
};
```

其中寄存器标识就是指代需要使用的寄存器，唯一需要注意的只有在处理mov指令时，还需要根据后一位的标志符来进行判断处理，这里就不再赘述。

以上的汇编代码其实是完全按照一段简单的加密翻译的，其实算法本身完全没有难度

```
unit8 szBuffer[] = "flag{w31c0m325wpuc7f@havea9o0d71m3}";
unit8 out[36];
int i = 0;

for (i = 0; i < 36; ++i) {
  if (i % 2) {
    out[i] = (szBuffer[i] + JI) ^ (0x68 - i);
  } else {
    out[i] = (szBuffer[i] - OU) ^ (0x67 + i);
  }
}

for (i = 0; i < 18; ++i) {
  out[i] ^= (i + 0x32);
  out[i + 18] ^= (i + 0x23);
}
```

所以我们甚至可以找到模拟的栈中存放我们输入的位置，并找到加密后的存放位置观察其变化，也能进行解析。

至于输入的位置其实可以看见就是利用了lea/rep movs两条指令，所以不管你输入多少，都会取出36位进行加密，只要找到这里，紧跟在后续的位置就是存放加密后的位置。

第三方WP

这一题可以说挺费耐心的，复现了三天左右才做出来。

首先

通过函数`sub_405bf0`实现其他操作的分发。

图中均为分发的函数，然后就是通过动态调试来查看这些函数都实现了怎样的功能。

其中比较重要的几个函数是`sub_402030`,`sub_402410`,`sub_4024c0`

在调试过程中会找到有几个地方的内存是作为寄存器使用，还有一个标志寄存器，不过对执行流程没有影响。

函数sub_4024c0,sub_402410对输入的数据进行加密，最后进行对比。加密的方式自己通过动态调试得到会更加清晰。

然后写脚本爆破得出flag

```
a="36 25 03 3C 25 28 65 6E 35 51 27 58 62 5E 41 6D 47 7C 6E 1A 63 18 04 02 34 03 9C 15 83 0C 9E 4F DC 4D C0 74"
b=bytearray.fromhex(a)
c1=list(b)
c2=[]
c3=[]
flag=""

for i in range(18):
    c=c1[i]^(0x32+i)
    c2.append(c)
for i in range(18):
    c=c1[18+i]^(0x23+i)
    c2.append(c)

d1=d2=0x67
for i in range(len(c1)):
    if i%2==0:
        c=c2[i]^(d1)
        c3.append(c)
        d1=d1+2
    else:
        c=c2[i]^(d2)
        c3.append(c)
        d2=d2-2

for i in range(36):
    if i%2==0:
        for j in range(48,127):
            v5=j^3
            k=3&(j^3)
            while(k):
                v2=2*k
                v5^=v2
                k=v2&v5
            if v5==c3[i]:
                flag+=chr(j)
    else:
        for jj in range(48,127):
            v8=jj^5
            kk=jj&5
            while(kk):
                v4=2*kk
                v55=v8
                v8^=v4
                kk=v4&v55
            if v8==c3[i]:
                flag+=chr(jj)

print(flag)
```

Flag

```
flag{w31c0m325wpuc7f@havea9o0d71m3}
```

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2020/SWPU-CTF-2020/Reverse/aWR9iTzjQ7QYZZiM1KnBB1.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # 2020 # Reverse # SWPU CTF 2020
where_is_temp
Re.exe