RE

vm

虚拟机保护，搜了搜资料，发现没啥好办法，只能一步一步动态调试。程序本身的逻辑十分简单，

```
int __cdecl main(int argc, const char **argv, const c
{
  FILE *v3; // rax
  int v4; // eax
  const char *v5; // rcx
  char input[40]; // [rsp+20h] [rbp-38h] BYREF

  puts("Welcome to ovm++!");
  sub_7FF6EB6E11EF();
  puts("Input your flag: ");
  v3 = _acrt_iob_func(0);
  fgets(input, 35, v3);
  printf("Your flag is: %s\n", input);
  puts("VM started successfully!");
  sub_7FF6EB6E128A((__int64)&build, (__int64)input);
  v4 = memcmp(input, &decoded, 0x22ui64);
  v5 = "nop";
  if ( !v4 )
    v5 = "good";
  puts(v5);
  return 0;
```

首先调用sub_7FF6EB6E11EF()显示vm的一些信息，不过我没看懂。。。函数的最后提到初始化寄存器，我也是从这里看出是虚拟机保护的。顺便命名一下：

```
JCQ1 = (__int64)result;
ebx = 0i64;
JCQ3 = 0;
JCQ4 = 0;
JCQ5 = 0;
```

，其中给JCQ1赋值的result在上面使用calloc来分配内存，

```
puts("setting VM's registers ......");
result = calloc(1ui64, 0x100ui64);
JCQ1 = (__int64)result;
ebx = 0i64;
JCQ3 = 0;
JCQ4 = 0;
JCQ5 = 0;
return result;
```

其实在虚拟机中是作为栈使用的。ebx原本是命名为JCQ2的，但是后来发现这个“寄存器”的作用就是储存输入的flag的地址，于是直接改名ebx。输入flag后调用sub_7FF6EB6E128A()来对flag进行加密，这个函数就是虚拟机保护的全部代码。于是针对这个函数进行动态调试。

尝试输入hgame{，来观察加密过程。首先进行的是case15,21,12,18,20,3,19，然后是21,12,18,20,3,19重复，其实就是对应以下汇编（其中一些写的简化了），作用是一个一个读取字符直到\0,也就是判断flag长度

```
15：push v5           ;v5就是IP
    mov v5,opdata
21：mov v16,input     ;按次序读入一个flag的字符
    push v16
12:pop JCQ4
18:cmp JCQ4,(char)JCQ3
20:je opdata          ;比较的是当前字符是否为\0,是\0就je
3 :BYTE1(JCQ3)+=opdata  ;实际运行时这里opdata是1,也就是说BYTE1(JCQ3)储存的是读取的字符数
20:jmp opdata         ;暂时不知道干啥的
```

然后如果cmp时发现JCQ4是\0,也就是到达flag最后一位,就执行以下case:11,12,16,8,13,18,20代码:

```
11:push BYTE1(JCQ3)      ;字符数入栈
12:pop JCQ4
16:pop and jmp           ;在这里是直接pop到了v5里,相当于jmp命令了
8 :push opdata           ;读取储存的opdata,就是储存的长度,34
13:pop LOBYTE(JCQ3)
18:cmp JCQ4,(char)JCQ3   ;比较字符数是否为34
20:je opdata             ;不相等就跳转,直接退出函数
```

通过以上翻译就得到以下信息:1.这个虚拟机程序先一个一个读取字符.2.JCQ3这个"寄存器"的HIBYTE(高8位)同步IP,BYTE2同步栈帧,BYTE1储存字符个数,BYTE一般作为比较.v5相当于IP,v9是栈底,v8是栈帧.

经过以上操作,得到flag长度为34,并且这个34包括\n.所以构造一个满足长度的flag:

```
hgame{12345678901234567890123456}
```

成功试出了加密流程:

先倒着逐个加密一遍:加密走的case是7,21,12,17,9,22,10,8,12,11,13,18,20,13,2,20

其实是做了一个异或,和一个初始数字是0xFE,每次加0x23的数字异或,并且由于是8位的无符号整型,所以它会加到上限0xFF然后回到0:

```
mov v16,输入(倒序)
push v16
pop JCQ4
JCQ4^=JCQ3           ;JCQ4与LOBYTE(JCQ3)异或
push JCQ4
pop v17
mov input,v17        ;修改字符
push JCQ3
push opdata
pop JCQ4
push BYTE1(JCQ3)
pop LOBYTE(JCQ3)
cmp JCQ4,(char)JCQ3 ;有没有到首字符
je opdata
pop LOBYTE(JCQ3)
je opdata(7)
add JCQ3,opdata
je opdata(不满足)
```

那么简单了,直接把主函数里memcmp函数里加密结束的密文异或一遍就可以了:

为啥不对!于是继续动态调试,原来还有一层加密过程...是将第一次加密过后的每个字符再减某数,某数是初始数字是0x7a,每次递减0x60,同样是个8位无符号整型.

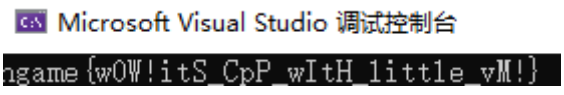
```
7 :BYTE1(JCQ3)-=opdata
21:mov v16,input(decoded)(倒序)
12:pop JCQ4
4 :     JCQ4-=JCQ3   (7A,1A,BA,5A)
9 :push JCQ4
22:pop v17 mov input,v17
10:push JCQ3
8 :push opdata
12:pop JCQ4
11:push BYTE1(JCQ3)
13:pop LOBYTE(JCQ3)
18:cmp JCQ4,(char)JCQ3
20:je opdata
13:pop LOBYTE(JCQ3)
6 :LOBYTE(JCQ3) = JCQ3 - opdata;
20:je opdata
```

于是写解密代码:

```
#include <stdio.h>
int main()
{
    int ch[] = {
0xCF,0x0BF,0x80,0x3B,0x0F6,0x0AF,0x7E,2,0x24,0x0ED,0x70,0x3A,0x0F4,0x0EB,0x7A,0x
4A,0x0E7,0x0F7,0x0A2,0x67,0x17,0x0F0,0x0C6,0x76,0x36,0x0E8,0x0AD,0x82,0x2E,0x0DB
,0x0B7,0x4F,0x0E6,9,9,0x16,0x2B,0x2D,0x42,0x44,0x4D };
    unsigned __int8 num = 0xFE;
    unsigned __int8 num2 = 0x7A;
    for (int i = 33; i >= 0; i--)
    {

        ch[i] = (ch[i] + num2) ^ num;
        num += 0x23;
        num2 -= 0x60;
    }
    for (int i = 0; i < 34; i++)
    {
        putchar(ch[i]);
    }
}
```

得flag

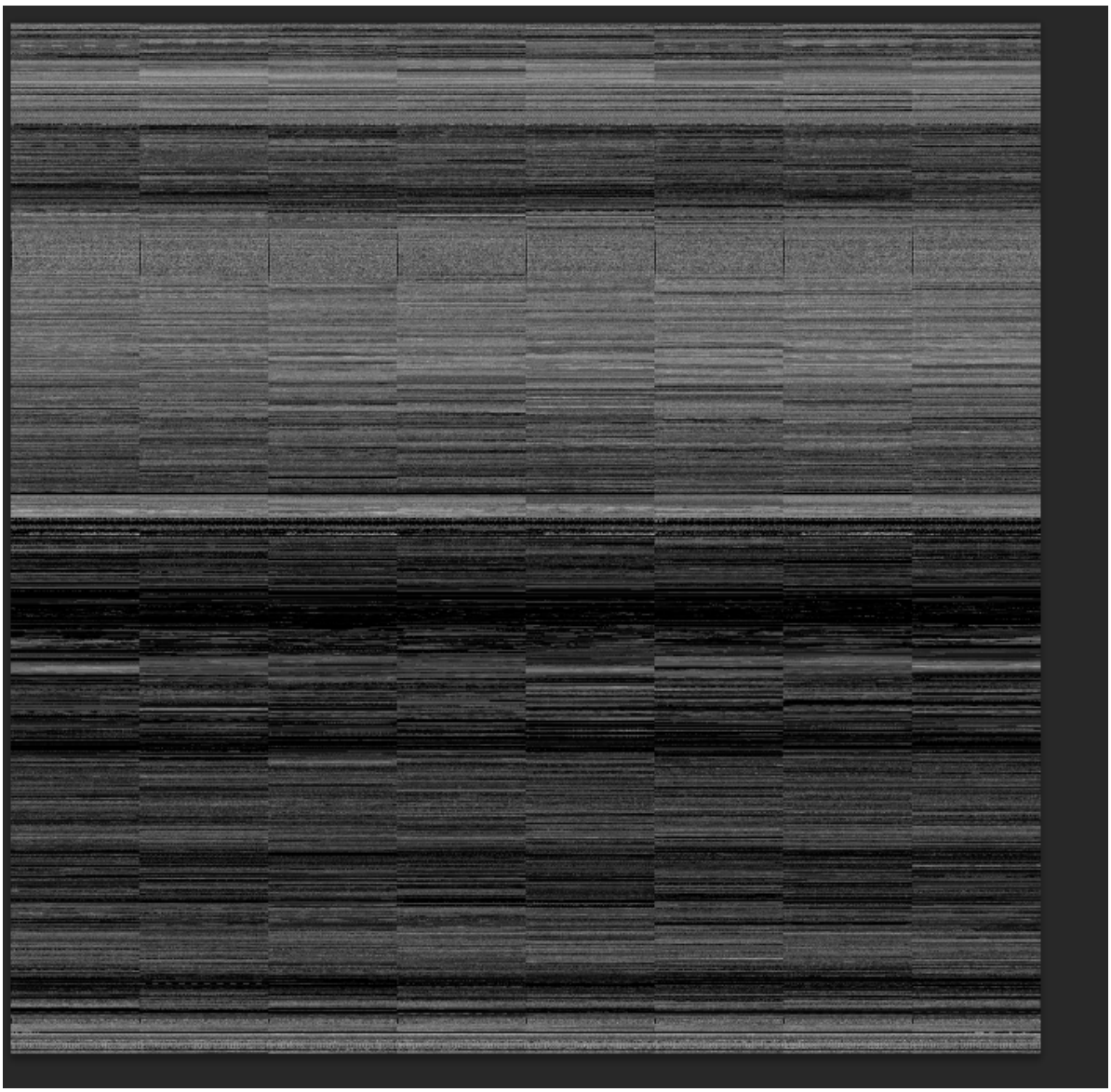


那么程序开始时显示的一堆东西是什么呢?我眼拙看不出来,不过猜是opcode和opdata,确实和程序中的操作数对应,但这个似乎没啥规律?要不就是我没看出来

misc

Akira之瞳-1

下载附件，解压，得一个RAW格式文件。以为是相机原图，于是用Photoshop打开：



emm这玩意可是不像啥东西。从网上了解到raw不光指图像的原数据，其他的也可以。最终得到这个是内存dump下来的，也就是得内存取证。于是找工具volatility。测profile知是win7x64内存镜像：

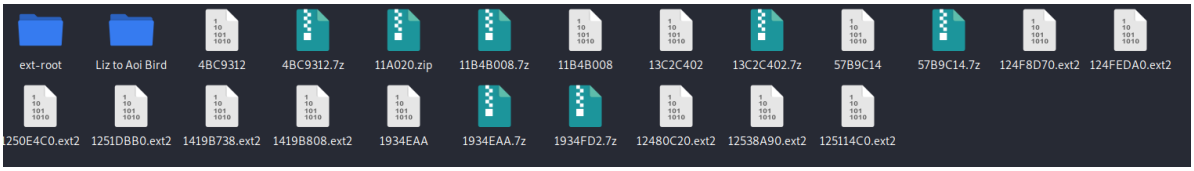
```
volatility_2.6_win64_standalone.exe imageinfo -f D:\important_work.raw
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64,
```

开始分析。

pslist命令列出运行的应用，得到important_work：

```
0xfffffa800f263b30 important_work        1092   2232      1       16      1      1 2021-02-18 09:47:15 UTC+0000
```

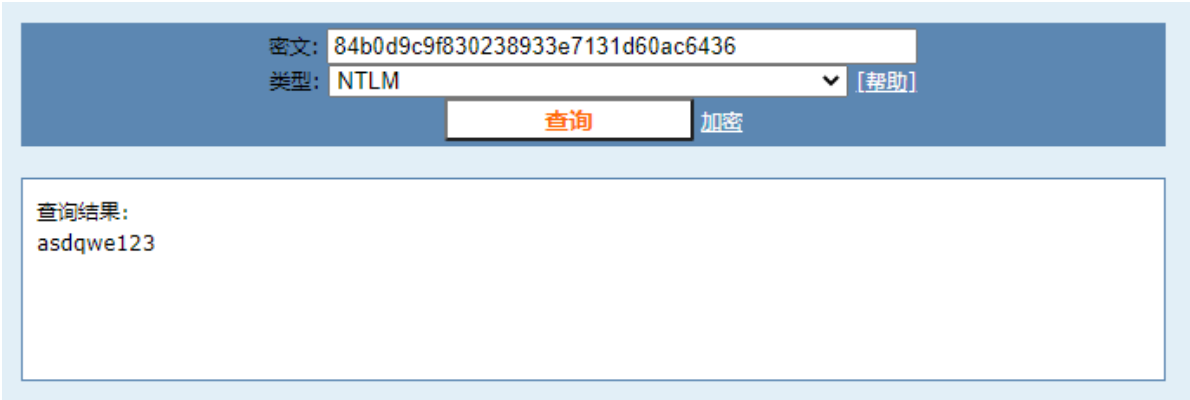
于是把这个东西dump下来，再扔到kali里面binwalk：



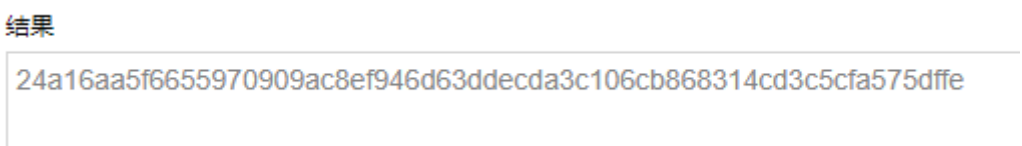
好家伙，这么多。。。但是binwalk自动“解压”出来一个Liz to Aoi Bird，这个应该就是可疑文件了，打开发现空文件，也就是说这个压缩包有密码了。于是找到压缩包11A020.zip查看，其注释写了：password is sha256(login_password),login_password是win登录密码？于是使用命令hivelist和hashdump得到以下数据：

```
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Genga03:1001:aad3b435b51404eeaad3b435b51404ee:84b0d9c9f830238933e7131d60ac6436:::
```

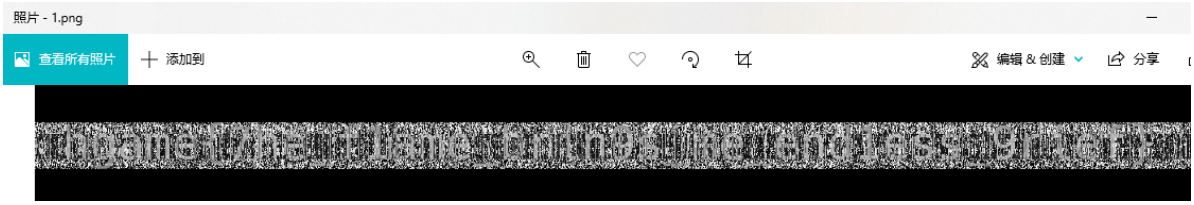
从网上得到win的登录密码是NTLM hash，于是hash在线破解：



再把这个密码用sha256加密，然后得到压缩包密码：



解压文件，得到两张一模一样的图片，两张图片的隐写方法挺多的，但是图片太大了，stegsolve卡死了，于是再去找工具，发现一个盲水印隐写，是两张图，于是下载试试，成功得到隐写内容：



这看的眼都要瞎了。。。Photoshop处理一下

奈斯。

总结

四周的hgame完了，也知道自己原来还是啥也不会，学那一点点，buu上做的那一点题，完全不够用。由于过年的一堆事情，week2有些题就没跟上做。虽然看wp复现了，但没有自己思考去做，和别人的差距就有点大。不过借这次竞赛，我也学到了很多，当时刚学re的时候连IDA都不知道，现在已经会简单的应用了，可惜自己的基础还是不够好，很多东西都是现学的。。。总之，无论是否能进协会，我都会在安全这条路上越钻越深！技术大于一切！