HGAME-Week4-Writeup

Re

1.vm

```
  sub_7FF6D87011EF();
  puts("Input your flag: ");
  v0 = _acrt_iob_func(0);
  fgets(Buffer, 35, v0);
  sub_7FF6D870114F("Your flag is: %s\n", Buffer);
  puts("VM started successfully!");
  sub_7FF6D870128A((__int64)&qword_7FF6D870E378, (__int64)Buffer);
  v1 = memcmp(Buffer, &unk_7FF6D870BD68, 0x22ui64);
  v2 = "nop";
  if ( !v1 )
    v2 = "good";
  puts(v2);
  return 0i64;
}
```

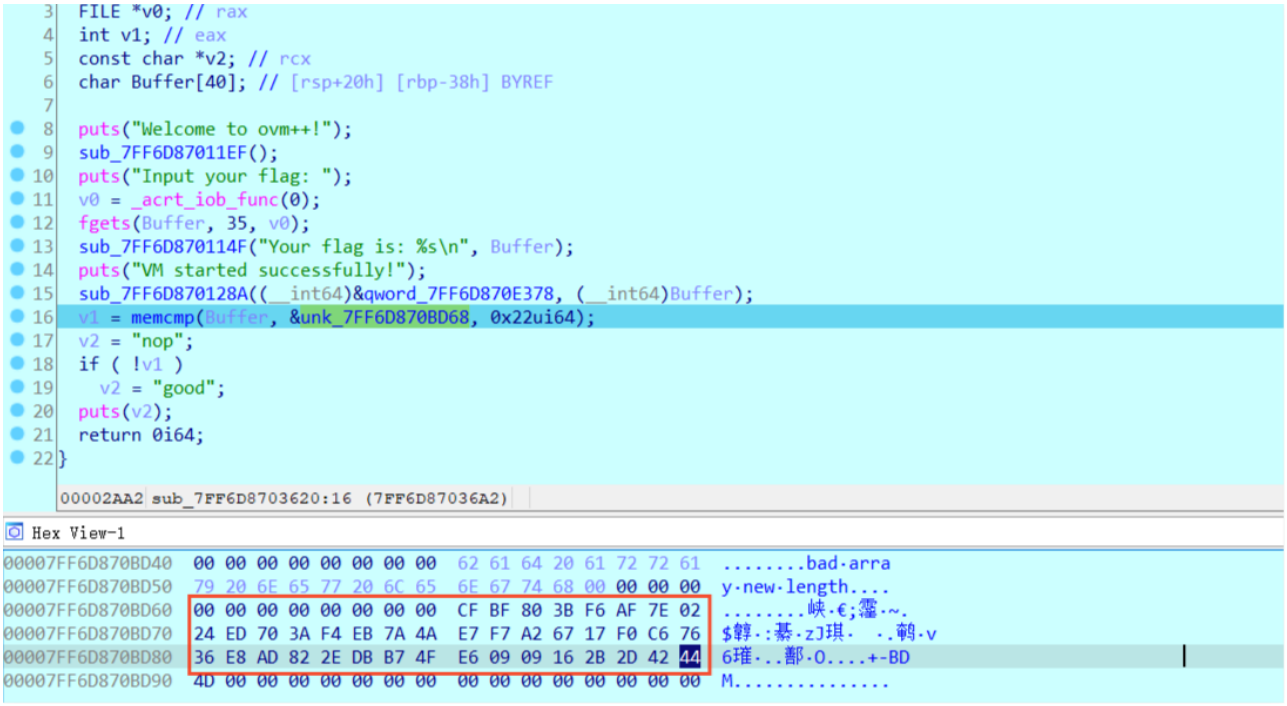
这部分逻辑很清晰，重点在中间这个加密的部分，点进去查看这个函数发现 switch-case 应该是vm，每个case都是一个 opcode

题目里说 ovm++ hates debugger ，那就先调试一下，调试后可以猜测出具体的加密部分

```
            BYTE2(dword_7FF6D870E3A9) = v8 + 1;
            *(_BYTE *)((unsigned __int8)(v8 + 1) + v9) = v5;
            v5 = v2;
            goto LABEL_33;
          case 16:
            v15 = v8--;
            v5 = *(_BYTE *)(v15 + v9);
            BYTE2(dword_7FF6D870E3A9) = v8;
            goto LABEL_34;
          case 17:
            byte_7FF6D870E3A8 ^= dword_7FF6D870E3A9;
            goto LABEL_34;
          case 18:
            byte_7FF6D870E3AD = byte_7FF6D870E3A8 == (char)dword_7FF6D870E3A9;
            goto LABEL_34;
          case 19:
            goto LABEL_28;
          case 20:
            if ( byte_7FF6D870E3AD )
```

```
            byte_7FF6D870E3A8 += v2;
            goto LABEL_34;
          case 2:
            LOBYTE(dword_7FF6D870E3A9) = v2 + dword_7FF6D870E3A9;
            goto LABEL_34;
          case 3:
            BYTE1(dword_7FF6D870E3A9) += v2;
            goto LABEL_34;
          case 4:
            byte_7FF6D870E3A8 -= dword_7FF6D870E3A9;
            goto LABEL_34;
          case 5:
            byte_7FF6D870E3A8 -= v2;
            goto LABEL_34;
          case 6:
            LOBYTE(dword_7FF6D870E3A9) = dword_7FF6D870E3A9 - v2;
            goto LABEL_34;
```

输入的 flag 先是经过异或加密，之后又有减法操作，加密后与密文进行比较，写脚本解一下就能得到 flag



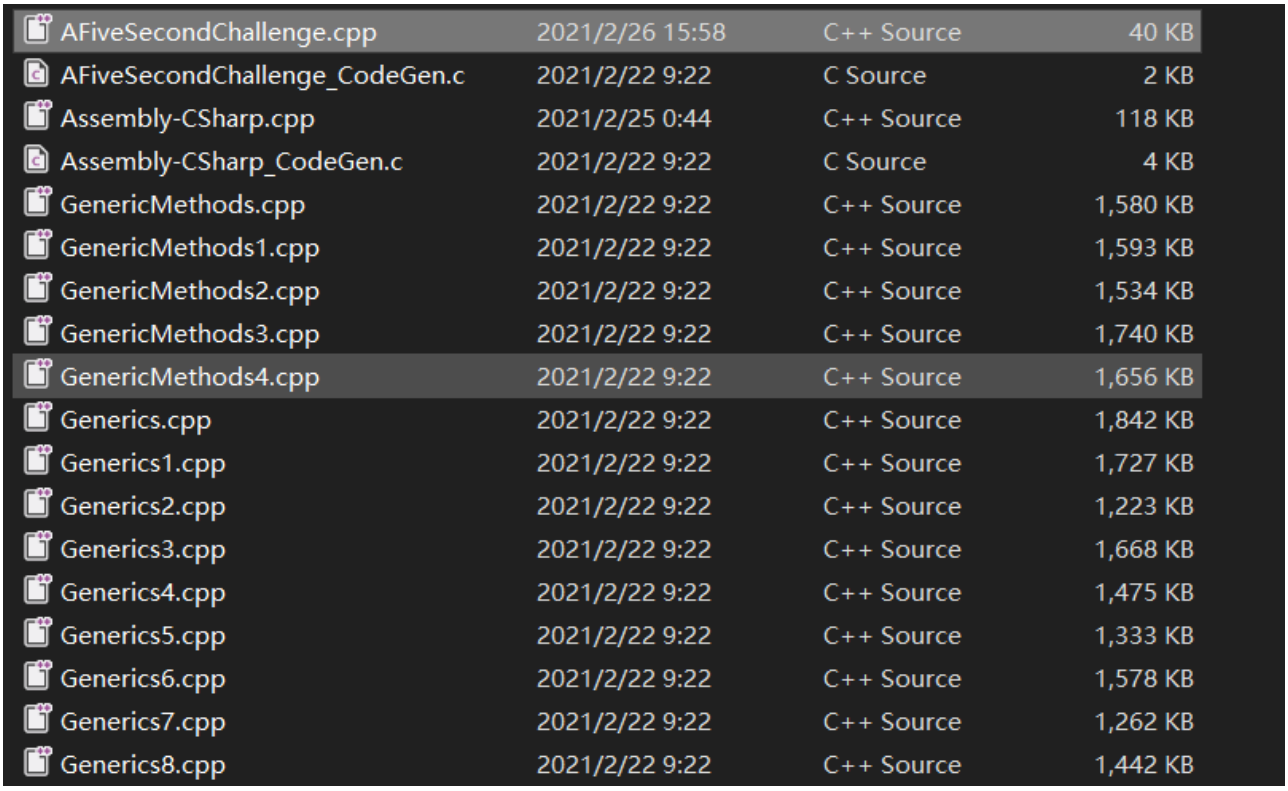

```
cipher =
[0xcf,0xbf,0x80,0x3b,0xf6,0xaf,0x7e,0x02,0x24,0xed,0x70,0x3a,0xf4,0
xeb,0x7a,0x4a,0xe7,0xf7,0xa2,0x67,0x17,0xf0,0xc6,0x76,0x36,0xe8,0xa
d,0x82,0x2e,0xdb,0xb7,0x4f,0xe6,0x09]
table1 =
[0xFE,0x21,0x44,0x67,0x8A,0xAD,0xD0,0xF3,0x16,0x39,0x5c,0x7f,0xa2,0
xc5,0xe8,0x0b,0x2e,0x51,0x74,0x97,0xba,0xdd,0x00,0x23,0x46,0x69,0x8
c,0xaf,0xd2,0xf5,0x18,0x3b,0x5e,0x81]
table2 =
[0x7a,0x1a,0xba,0x5a,0xfa,0x9a,0x3a,0xda,0x7a,0x1a,0xba,0x5a,0xfa,0
x9a,0x3a,0xda,0x7a,0x1a,0xba,0x5a,0xfa,0x9a,0x3a,0xda,0x7a,0x1a,0xb
a,0x5a,0xfa,0x9a,0x3a,0xda,0x7a,0x1a]

for i in range(34):
    if cipher[i] + table2[33-i] < 256:
        cipher[i] += table2[33-i]
    else:
        cipher[i] = cipher[i] + table2[33-i] - 256

for i in range(34):
    cipher[i] ^= table1[33-i]
    print(chr(cipher[i]),end = "")
```

这道题标准解法应该不是这样解的，而是要先要翻译出对应的汇编代码再分析，但这部分我还不太明白之后再补上吧

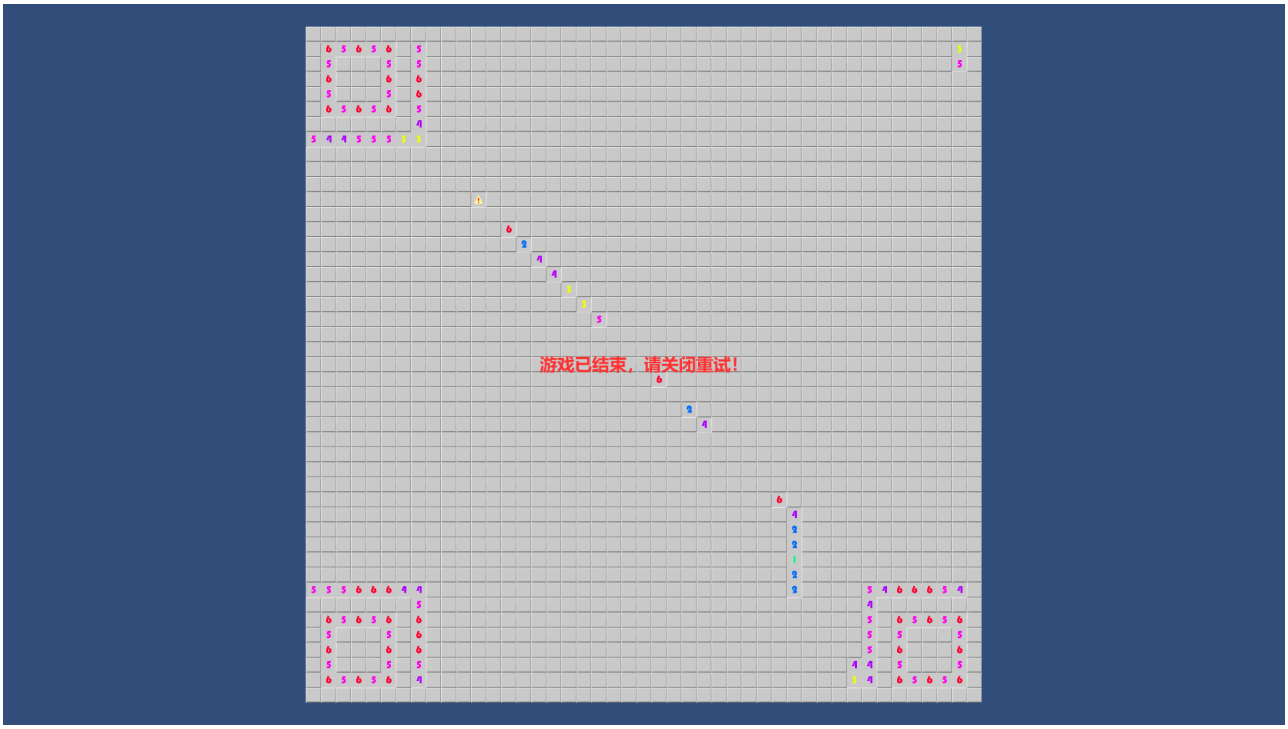
2.A 5 Second Challenge

这题要感谢小圆学长的耐心回答!!

下载得到的是用unity制作的扫雷游戏，刚开始为了获得源码用 il2cppdumper 处理 GameAssembly.dll ，但发现处理后得到的不全，之后问了一下小圆学长发现源码已经给了。



打开 AFiveSecondChallenge.cpp ，源码里有 getUnixtime 这类获取时间的函数，游戏时间超过 5s 就会显示超时，用锁住系统时间的软件可以避开这一检测。锁定时间后就可以开始玩扫雷，几局下来后会发现雷的位置能构成二维码



根据题目提示，把 managed 文件夹下的 dll 拖到 ida 里，会发现有一个函数被nop掉了，定位到源码里查看（il2cpp的中间文件有些混乱，删除了影响判断的部分）

```
NullCheck(L_3);
double L_8 = (L_3)->GetAt((L_5), (L_7) / 3), 0);
L_9 = (il2cpp_codegen_static_fields_for(BombChecker_il2cpp_TypeInfo_var))->get_matrix_6();
L_10 = ___vec0;
float L_11 = L_10.get_y_1();
L_12 = ___vec0;
float L_13 = L_12.get_x_0();
NullCheck(L_9);
double L_14 = (L_9)->GetAt(L_11, L_13 / 3, 1);
V_0 = L_14;
L_15 = (il2cpp_codegen_static_fields_for(BombChecker_il2cpp_TypeInfo_var))->get_matrix_6();
L_16 = ___vec0;
float L_17 = L_16.get_y_1();
L_18 = ___vec0;
float L_19 = L_18.get_x_0();
NullCheck(L_15);
double L_20 = (L_15)->GetAt(L_17, (L_19) / 3, 2);
V_1 = L_20;
L_21 = ___vec0;
float L_22 = L_21.get_x_0();
V_2 = fmodf(L_22, 3.0f) - 1.0f;
double L_23 = V_2;
double L_24 = V_2;
double L_25 = V_0;
double L_26 = V_2;
double L_27 = V_1;
return (((il2cpp_codegen_add((il2cpp_codegen_add((il2cpp_codegen_multiply((il2cpp_codegen_multiply(L_8,
L_8 * L_23 * L_24 + L_25 * L_26 + L_27 > 0.0 ? 1 : 0
```

这里的 return 判断是否是雷的位置，本来分析到这里已经很清楚了但是因为我完全不懂类和方法什么的看了挺久

```
inline double GetAt(i,j,k) const
    {
        iBound = bounds[0].length;  //45
        IL2CPP_ARRAY_BOUNDS_CHECK(i, iBound);
        jBound = bounds[1].length;  //15
        IL2CPP_ARRAY_BOUNDS_CHECK(j, jBound);
        kBound = bounds[2].length;  //3
        IL2CPP_ARRAY_BOUNDS_CHECK(k, kBound);

        index = (i * jBound + j) * kBound + k;
        return m_Items[index];
    }
IL_001c:
```

GetAt函数的返回值没在源码里看到有初始赋值于是迷惑了好久，又去问了一下小圆学长这部分是在哪里初始化过，得到回复说是在那个 dll 里面。

但是我在我下载的东西里面没发现 m_Items 初始化的部分，去网站上重新下载并拖到 ilspy ，得到了这个数组里的值

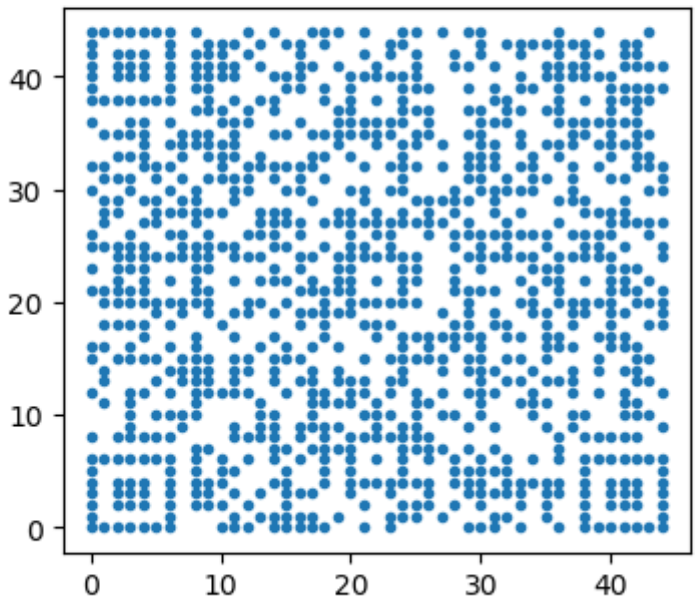
```
    private static double[,,] matrix = new double[45, 15, 3]
    {
        {
            { 0.286404298659526, -0.614010617480656, 0.340978330362628 },
            { 0.1633000915867, -0.01792766798079, 1.72356132301133 },
            { 2.32664311949988, -1.21263336242711, -1.58544931499367 },
            { -0.89197766680511, 0.586576085816987, 0.826835871060768 },
            { 1.23022195645206, -0.13595393417673, -0.828730780800369 },
            { 0.537928296734488, 0.388014349598005, 0.0374356689733466 },
            { 0.814202140660664, -0.99166360620885, -0.737111428069284 },
            { 2.00547921600026, -0.147569149330607, -1.04956539635377 },
            { 0.1821082206676, -0.28397895566802, -1.7011248571328 },
            { 1.21535750767622, 0.224098292168243, -1.27386082038091 },
            { -1.50304427906392, -1.03580711300699, 1.02076515571834 },
            { 1.49346530835591, -0.725015526868452, -1.85215913935819 },
            { 3.31221117959322, -0.40291529800406, -1.86138486321435 },
            { 0.716296624704974, 0.235190708626375, 0.700813831687661 },
            { -0.493886018132696, -0.446160607591385, 1.8801513313814 }
        },
        {
            { 1.64912200745308, -0.891905981800886, -1.5080679911496 },
            { 0.535460775094064, -0.345045054339812, -1.07124592803708 },
            { 1.72166307770579, -0.39340525532453, -1.66794063721442 },
            { 0.589413534133468, 1.14779187518606, -0.545213267244733 },
            { -0.962039449926566, 1.30072123048117, 0.318279494232945 },
            { 0.497930590669345, 0.437612881218065, 1.01194782651236 }
```

接下来就是写脚本解题

```
#include<stdio.h>
#include<math.h>

double matrix[45][15][3] = ;

double GetAt(i, j, k)
{
    int iBound = 45;    //45
    int jBound = 15;    //15
    int kBound = 3; //3
    return matrix[i][j][k];
}
int main()
{
    int y,x;
    FILE* fp;
    fp = fopen("************" , "w");
    int a[45][45];
    int i, j;
    for (y = 0; y < 45; y++)
    {
        for (x = 0; x < 45; x++)
        {
            double L_8 = GetAt(y, x / 3, 0);
            double L_14 = GetAt(y, x / 3, 1);
            double V_0 = L_14;
            double L_20 = GetAt(y, x / 3, 2);
            double V_1 = L_20;
            double V_2 = fmod(x, 3.0f) - 1.0f;
            a[x][y] = L_8* V_2* V_2 + V_0 * V_2 + V_1 > 0.0 ? 1 :
0;    //1是雷
        }
    }

    for (i = 0; i < 45; i++)
    {
        for (j = 0; j < 45; j++)
        {
            printf("%d", a[j][i]);
            if (a[j][i] == 1)
                fprintf(fp, "%d,%d\n", j, i);
        }
        printf("\n");
```

```
    }
    fclose(fp);
    return 0;
}
```

```
import matplotlib.pyplot as plt
import numpy as np

x,y = np.loadtxt('./xy.txt',delimiter=',',unpack=True)
plt.plot(x,y,'.')
plt.show()
```



3.nllvm

拖到 ida 里发现挺乱的，也没看到控制台输出的那些文字，那就先调试看看

```
byte_7FF78B121286 ^= 0x82u;
byte_7FF78B121287 ^= 0xE5u;
byte_7FF78B121288 ^= 0xE6u;
byte_7FF78B121289 ^= 0x5Bu;
byte_7FF78B12128A ^= 0xC4u;
byte_7FF78B12128B ^= 0xA1u;
byte_7FF78B12128C ^= 0x6Fu;
byte_7FF78B12128D ^= 0xC1u;
byte_7FF78B12128E ^= 0xC8u;
byte_7FF78B12128F ^= 0x2Cu;
byte_7FF78B121290 ^= 0x42u;
byte_7FF78B121291 ^= 0x2Du;
byte_7FF78B121292 ^= 0x18u;
byte_7FF78B121293 ^= 0xBFu;
byte_7FF78B121294 ^= 0x43u;
byte_7FF78B121295 ^= 0x3Fu;
byte_7FF78B121296 ^= 0xE3u;
byte_7FF78B121297 ^= 0x9Au;
byte_7FF78B121298 ^= 0x41u;
byte_7FF78B121299 ^= 0xD9u;
byte_7FF78B12129A ^= 0x95u;
byte_7FF78B12129B ^= 0x47u;
byte_7FF78B12129C ^= 0x2Cu;
byte_7FF78B12129D ^= 0x80u;
byte_7FF78B12129E ^= 0xABu;
byte_7FF78B12129F ^= 0x22u;
byte_7FF78B1212A0 ^= 0x82u;
byte_7FF78B1212A1 ^= 5u;
v4 = sub_7FF78B0E7AE0(&off_7FF78B121730, &byte_7FF78B121281);
byte_7FF78B121281 ^= 0xCEu;
byte_7FF78B121282 ^= 0xD1u;
byte_7FF78B121283 ^= 0xC1u;
```

调试后发现这些异或操作得到的就是控制台输出的字符，在输出后又来了一遍异或还原。

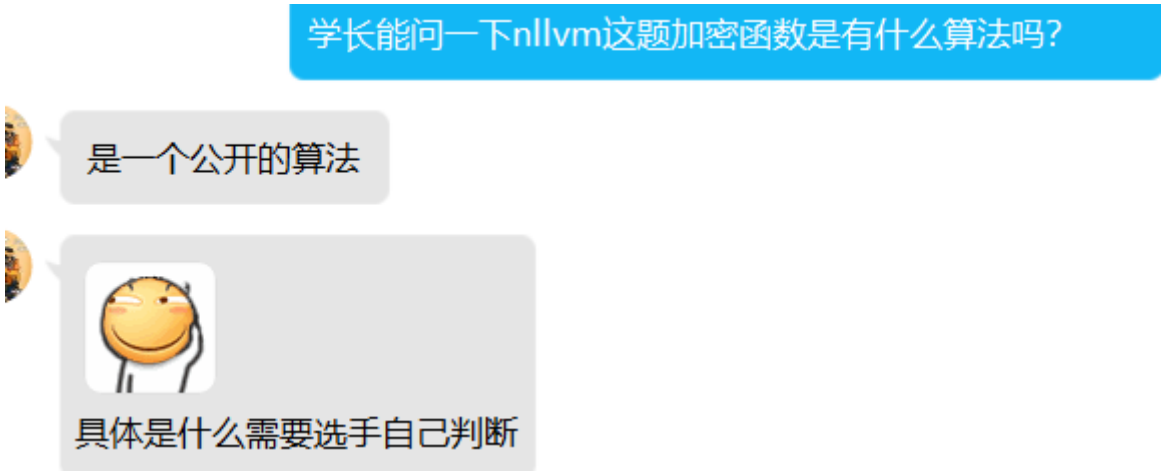
接着调试发现判断输入字符串的地方

```
464  memset(Str, 0, sizeof(Str));
465  sub_7FF78B0E7EA0(&off_7FF78B1217A0, Str);
466  if ( strlen(Str) == 64 )
467  {
468    v18[0] = 0x706050403020100i64;
469    v18[1] = 0xF0E0D0C0B0A0908i64;
470    memset(Buf2, 0, sizeof(Buf2));
471    memmove(Buf2, Str, 0x40ui64);
472    byte_7FF78B121210 ^= 0xCEu;
473    byte_7FF78B121211 ^= 0xD1u;
```

这一部分异或得到 **“CryptoFAILUREforRSA2048Key!!!!!!”** 这个字符串，接着来到加密的函数

```
523     byte_7FF78B121221 ^= 0x18u;
524     byte_7FF78B121222 ^= 0xBFu;
525     byte_7FF78B121223 ^= 0x43u;
526     byte_7FF78B121224 ^= 0x3Fu;
527     byte_7FF78B121225 ^= 0xE3u;
528     byte_7FF78B121226 ^= 0x9Au;
529     byte_7FF78B121227 ^= 0x41u;
530     byte_7FF78B121228 ^= 0xD9u;
531     byte_7FF78B121229 ^= 0x95u;
532     byte_7FF78B12122A ^= 0x47u;
533     byte_7FF78B12122B ^= 0x2Cu;
534     byte_7FF78B12122C ^= 0x80u;
535     byte_7FF78B12122D ^= 0xABu;
536     byte_7FF78B12122E ^= 0x22u;
537     byte_7FF78B12122F ^= 0x82u;
538     byte_7FF78B121230 ^= 5u;
539     encrypt((__int64)v16, Buf2, 0x40ui64);
540     byte_7FF78B121240 ^= 0xCEu;
541     byte_7FF78B121241 ^= 0xD1u;
542     byte_7FF78B121242 ^= 0xC1u;
543     byte_7FF78B121243 ^= 0xB6u;
544     byte_7FF78B121244 ^= 0x78u;
545     byte_7FF78B121245 ^= 0x82u;
546     byte_7FF78B121246 ^= 0xE5u;
547     byte_7FF78B121247 ^= 0xE6u;
548     byte_7FF78B121248 ^= 0x5Bu;
549     byte_7FF78B121249 ^= 0xC4u;
550     byte_7FF78B12124A ^= 0xA1u;
551     byte_7FF78B12124B ^= 0x6Fu;
552     byte_7FF78B12124C ^= 0xC1u;
553     byte_7FF78B12124D ^= 0xC8u;
```

传入的就是这部分异或得到字符串和输入的 flag，之后看了一下这个函数里面是一些比较复杂的运算猜测是某种算法，我对各种算法不是很熟所以用了 ida 插件 findcrypt ，但是这个插件并没有找到什么算法于是我就去找学长确认了一下



查了各种算法，最后发现加密函数里面的一个函数像是**s盒**，根据s盒数字判断又是 **AES 加密**（s盒也是异或得到的，这大概是插件检测不出算法的原因）

```
{
  __int64 result; // rax
  __int64 v2; // rcx
  char v3; // dl
  unsigned __int8 j; // [rsp+6h] [rbp-Ah]
  unsigned __int8 i; // [rsp+7h] [rbp-9h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= 4u )
      break;
    for ( j = 0; j < 4u; ++j )                  // AES s盒
    {
      v2 = *(unsigned __int8 *)(4i64 * j + a1 + i);
      byte_7FF78B121000[0] ^= 0xCEu;            // 63
      byte_7FF78B121001 ^= 0xD1u;               // 7c
      byte_7FF78B121002 ^= 0xC1u;
      byte_7FF78B121003 ^= 0xB6u;
      byte_7FF78B121004 ^= 0x78u;
      byte_7FF78B121005 ^= 0x82u;
      byte_7FF78B121006 ^= 0xE5u;
      byte_7FF78B121007 ^= 0xE6u;
      byte_7FF78B121008 ^= 0x5Bu;
      byte_7FF78B121009 ^= 0xC4u;
      byte_7FF78B12100A ^= 0xA1u;
      byte_7FF78B12100B ^= 0x6Fu;
      byte_7FF78B12100C ^= 0xC1u;
      byte_7FF78B12100D ^= 0xC8u;
      byte_7FF78B12100E ^= 0x2Cu;
      byte_7FF78B12100F ^= 0x42u;
```

在调试过程中可以知道 **iv值是 123456789abcdef**，判断是否为正确 flag 的部分与密文进行比较，改变了输出字符的外观（表示是否回到现实）

```
pwn god comsos one shot one flag
vegetable bird Y coding with his girl
        world line through a "pass"
        something fucking happen
we have not seen cosmos pwn anymore
could you help cosmos back to life
```

之后用我在week2写的脚本解就行

```
解密后:hgame{cOsm0s_is_still_fight1ng_and_NEVER_GIVE_UP_O0o0o0oO00o00o}

Process finished with exit code 0
```

查了一下题目里的 llvm，是一款代码混淆器，具体的是看了这篇文章https://bbs.pediy.com/thread-224484.htm

Misc

1.Akira之瞳-1

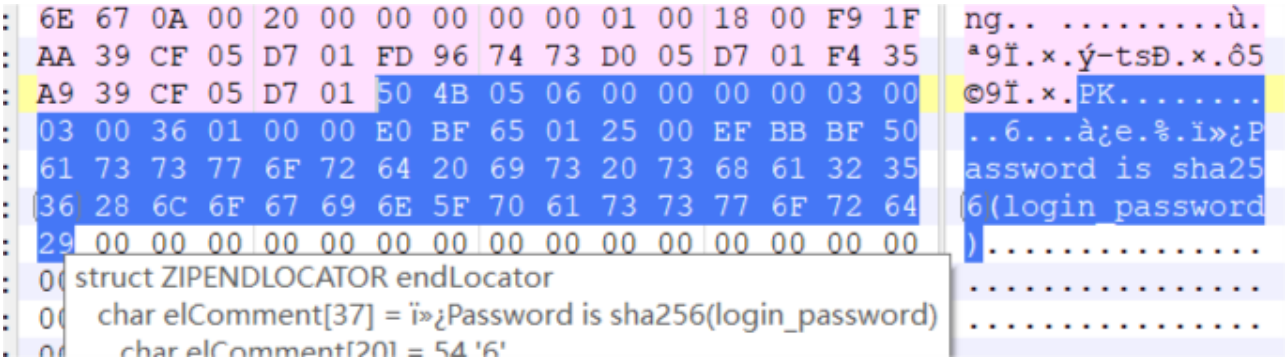
查资料知道这种 dump 出来的RAW 文件要在 linux 里用 **volatility** 看，先查一下系统版本

```
Desktop volatility -f important_work.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...

          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win200
8R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/linux/Desktop/important
_work.raw)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf8000403b0a0L
          Number of Processors : 16
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff8000403cd00L
                KPCR for CPU 1 : 0xfffff88004700000L
                KPCR for CPU 2 : 0xfffff88004776000L
                KPCR for CPU 3 : 0xfffff880047ec000L
                KPCR for CPU 4 : 0xfffff88004840000L
                KPCR for CPU 5 : 0xfffff880048b6000L
                KPCR for CPU 6 : 0xfffff8800492c000L
                KPCR for CPU 7 : 0xfffff880049a2000L
                KPCR for CPU 8 : 0xfffff880049d8000L
                KPCR for CPU 9 : 0xfffff88004a94000L
               KPCR for CPU 10 : 0xfffff88004b0a000L
               KPCR for CPU 11 : 0xfffff88004b80000L
```

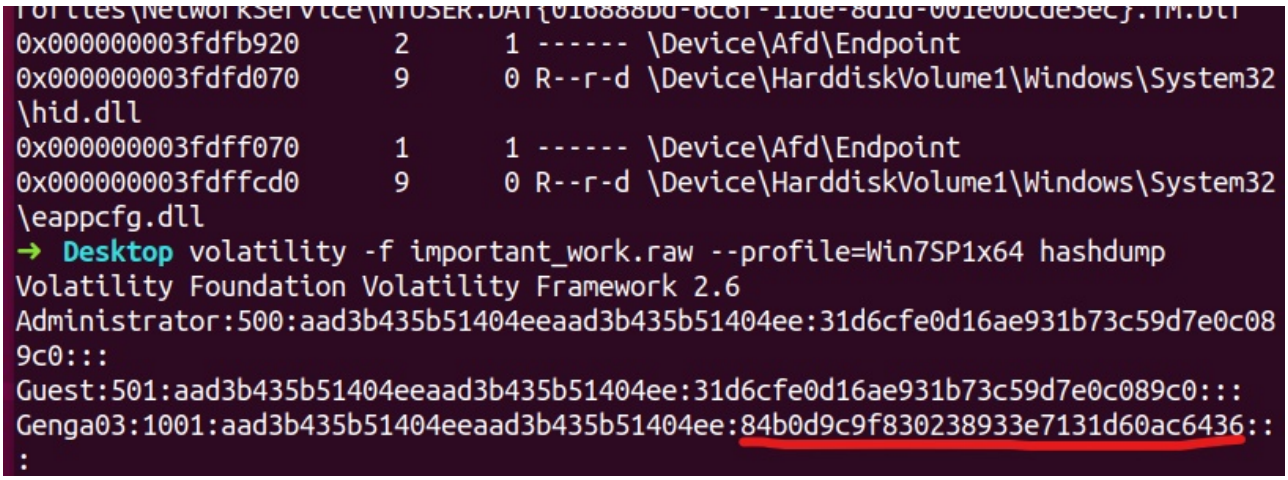
接着用 pslist 查看一下系统进程，找到可疑的进程后 dump 出来

```
0xfffffa800ec09b30 dwm.exe                    1340    940      7      15
0xfffffa800ec12b30 explorer.exe               2232   3064     32      71
0xfffffa800ecaf210 vm3dservice.ex             1364   2232      5       8
0xfffffa800ec313e0 vmtoolsd.exe               1268   2232      9      18
0xfffffa800e5ab460 taskmgr.exe                2780    680     12      14
0xfffffa800e5c6b30 SearchIndexer.             1252    568     13      64
0xfffffa800ed50b30 wmpnetwk.exe               2572    568     13      25
0xfffffa800ed2eb30 svchost.exe                2596    568     13      18
0xfffffa800f246670 SearchProtocol              736   1252      7      24
0xfffffa800f248060 SearchFilterHo             2552   1252      5      10
0xfffffa800f263b30 important_work             1092   2232      1       1
0xfffffa800f260060 conhost.exe                1372    520      2       6
0xfffffa800f29fb30 cmd.exe                    1340   1092      1       2
0xfffffa800ec13590 dllhost.exe                3128    720      6      10
```

用 foremost 分离出一个加密的压缩包，本来以为密码还在 raw 文件里面的某个文件里来回看了好多遍，之后用 010editor 打开压缩包后发现提示，zip 密码是**系统登陆密码**



hashdump 一下得到登陆密码的 hash 值



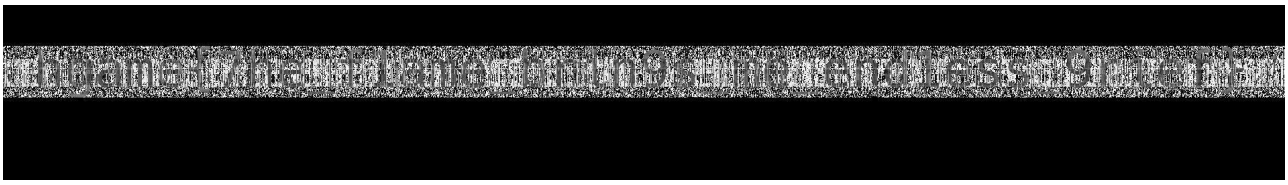
按提示把密码解出来

asdqwe123

在线加密 在线解密

sha256（asdqwe123）= 20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e91bca4c3

打开压缩包后发现两张图片，名字是src 和 blind，很容易就想到盲水印，用这个工具 (https://github.com/chishaxie/BlindWaterMark) 提取出水印



图片里小写的 L 是数字1

2.Akira之瞳-2

得到一个加密压缩包和 RAW 文件，同样先查看一下进程，发现 notepad进程 dump出来

```
0xfffffa801b13db00 explorer.exe          2372   2580     29      854      1
  0 2021-02-19 08:06:46 UTC+0000
0xfffffa801babfb00 vm3dservice.ex        2792   2372      2       79      1
  0 2021-02-19 08:06:48 UTC+0000
0xfffffa801bac8b00 vmtoolsd.exe          2944   2372      8      232      1
  0 2021-02-19 08:06:48 UTC+0000
0xfffffa801bb56b00 SearchIndexer.        1308    560     12      676      0
  0 2021-02-19 08:06:54 UTC+0000
0xfffffa801b14db00 wmpnetwk.exe          1384    560      9      240      0
  0 2021-02-19 08:06:54 UTC+0000
0xfffffa801bb7b730 svchost.exe           3104    560     13      262      0
  0 2021-02-19 08:06:55 UTC+0000
0xfffffa801b9f7470 svchost.exe           4020    560     12      379      0
  0 2021-02-19 08:08:12 UTC+0000
0xfffffa801b475b00 notepad.exe            456   2372      1       63      1
  0 2021-02-19 08:19:52 UTC+0000
0xfffffa801a154060 SearchProtocol        3732   1308      8      294      0
  0 2021-02-19 08:22:20 UTC+0000
0xfffffa801a175b00 SearchFilterHo        2080   1308      5      118      0
  0 2021-02-19 08:22:20 UTC+0000
0xfffffa801b172060 chrome.exe            3948   2372     43      815      1
  0 2021-02-19 08:22:27 UTC+0000
0xfffffa801b15e060 chrome.exe            4052   3948      8       92      1
```

在 dump 出来的进程里查一下 password 字符串，得到 zip 密码

```
→ Desktop strings -e l 456.dmp | grep password
zip password is: 5trqES&P43#y&1TO
Automatic logon with current user name and password
)&&-1==i.indexOf("$")&&(i="^(?:"+i+")$"),fieldcacheset(e,n,"input_pattern",i),i)
:null}function LP_get_input_pattern_desc(e,t){if(!e||!t)return null;var n=t.getA
```

打开压缩包得到

名称	大小	压缩后大小	修改时间	属性	CRC	加密	算法
S-1-5-21-26271...	468	0	2021-02-1...	SD	844D6A3B	-	
container	10 485 760	10 487 888	2021-02-1...	A	2E0913D8	+	LZMA2:12...
Cookies	20 480		2021-02-1...	A	5B0B396D	+	LZMA2:12...

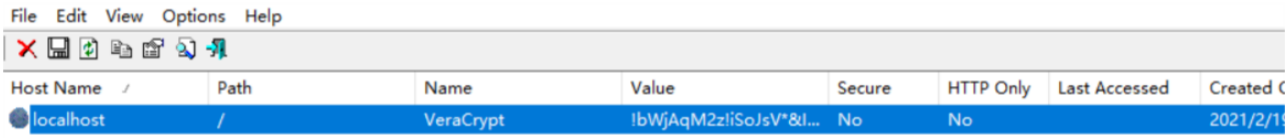
这几个文件名好像在哪里见过，翻了一下往年的 writeup 知道container 是**加密容器**，cookies 是chrome浏览器的 **cookie 数据库**。但是看 cookies 文件内容和打开加密卷都需要密码，回到 linux 里再查一下有无密码，因为是 notepad 进程那就查 **txt** 文件

```
0x000000007ef94820      2      0 RW-r-- \Device\HarddiskVolume1\Users\Genga03\De
sktop\dumpme.txt
0x000000007efbbc00      1      1 -W-rw- \Device\HarddiskVolume1\Users\Genga03\Ap
pData\Local\Temp\FXSAPIDebugLogFile.txt
0x000000007f26d330      1      0 R--r-- \Device\HarddiskVolume1\Program Files\7-
Zip\Lang\zh-cn.txt
0x000000007f2b5f20      2      0 RW-rw- \Device\HarddiskVolume1\Users\Genga03\Ap
pData\Roaming\Microsoft\Windows\Recent\dumpme.txt.lnk
0x000000007f497ea0      2      0 RW-rw- \Device\HarddiskVolume1\Users\Genga03\Ap
pData\Roaming\Microsoft\Windows\Recent\cmp2.txt.lnk
0x000000007f5be180     16      0 R--rwd \Device\HarddiskVolume1\ProgramData\VMwa
re\VMware Tools\manifest.txt
→ Desktop volatility -f secret_work.raw --profile=Win7SP1x64 dumpfiles -Q 0x000
000007ef94820 -D ./
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7ef94820   None   \Device\HarddiskVolume1\Users\Genga03\Desk
top\dumpme.txt
→ Desktop cat file.None.0xfffffa801aa35340.txt 4
zip password is: 5trqES&P43#y&1TO
And you may need LastPasscat: 4: 没有那个文件或目录
→ Desktop cat file.None.0xfffffa801aa35340.txt
zip password is: 5trqES&P43#y&1TO
And you may need LastPass%
→ Desktop
```

这个 **dumpme.txt** 很可疑，提取出来发现之前 zip 密码还有后半句。百度一下知道 lastpass 是 chrome 浏览器的插件，用来记录密码的，把 lastpass 和 dump 放在一起搜索找到 volatility 里有 lastpass 这个插件

```
→ Desktop volatility --plugins=/usr/share/volatility/contrib/plugins/lastpass --profile=Win7SP1x64 -f ./secret_work.raw lastpass
Volatility Foundation Volatility Framework 2.6
Searching for LastPass Signatures
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)

Found LastPass Entry for live.com
UserName: windows login & miscrosoft
Pasword: Unknown

Found LastPass Entry for live.com,bing.com,hotmail.com,live.com,microsoft.com,msn.com,windows.com,windowsazure.com,office.com,skype.
UserName: windows login & miscrosoft
Pasword: vIg*q3x6GFa5aFBA
```

得到一个密码，下载 **ChromeCookiesView**，加载 cookies 文件并输入密码，得到加密容器密码并提示用 **VeraCrypt** 打开



打开虚拟分区得到一张图片，查 ADS 得知 NTFS 隐写

> *NTFS交换数据流（Alternate Data Streams，简称ADS）是NTFS磁盘格式的一个特性。在NTFS文件系统下，每个文件都可以存在多个数据流，意思是除了主文件流之外还可以有许多非主文件流寄宿在主文件流中，这些利用NTFS数据流寄宿并隐藏在系统中的非主文件流我们称之为ADS流文件。虽然我们无法看到ADS流文件，但它们却是真实存在。*

然后用 **Ntfs Streams Editor** 这个软件查看 ADS 流文件，得到 flag

那个网址里的图是 Akira 学长的新头像

总结

为时四周的 hgme 要结束了，这个寒假学到了很多东西过得很充实，希望以后能继续学习这方面的知识，不断进步