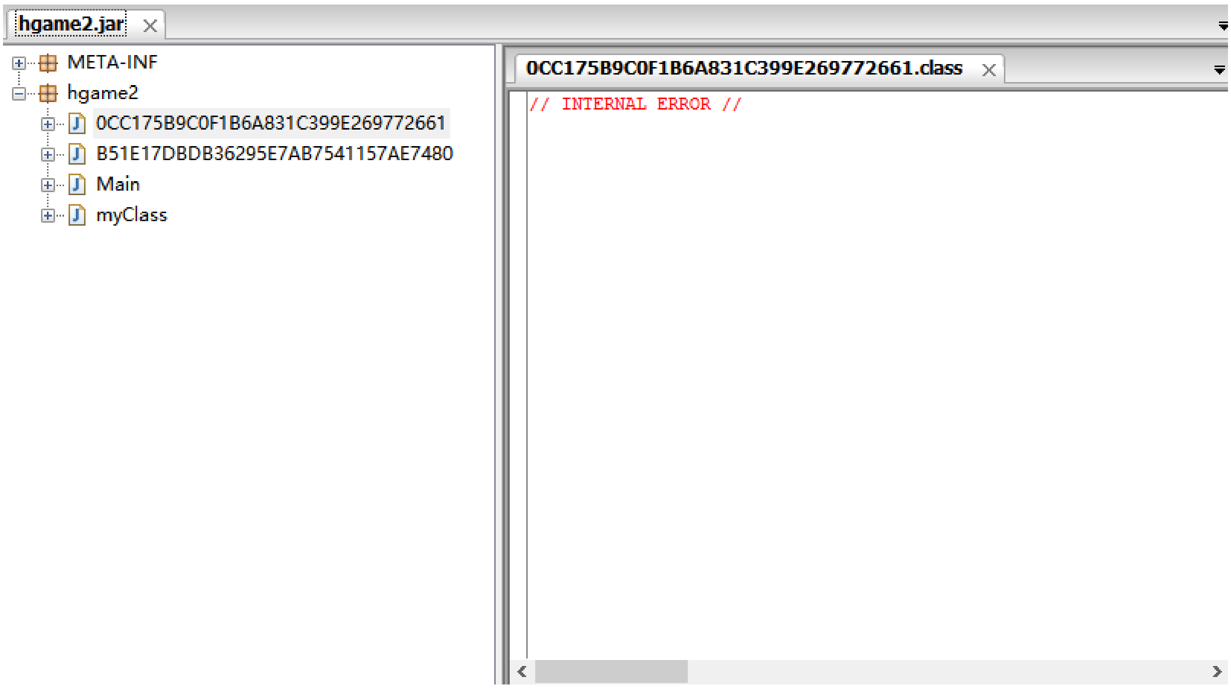
Week4 WP - Veritas501

RE

explorer的奇怪番外4

做完以后我其实想说一句，这题真的不难，是我太蠢。

首先掏出jd-gui，看到这个jar由两个加密的class和两个没加密的class组成：


hgame2.jar
META-INF
hgame2
0CC175B9C0F1B6A831C399E269772661
B51E17DBDB36295E7AB7541157AE7480
Main
myClass
0CC175B9C0F1B6A831C399E269772661.class

```
// INTERNAL ERROR //
```

jar里完整的源码我就不放了，看到两段：

```
{
    myClass mc = new myClass();
    Class clazz = mc.loadClass("hgame2.checkFlag");
    Method c = clazz.getMethod("trueMain", (Class[])null);
    c.invoke(null, new Object[0]);
}
```

```
try {
      res = res.substring(0, l + 1) + md5(className.getBytes()) + ".class";
} catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
}
```

得知，以上加密的两个class的名字是原来名字的md5值，由代码知一个为`checkFlag`，MD5破解得另一个为`a`，当然这不是重点。

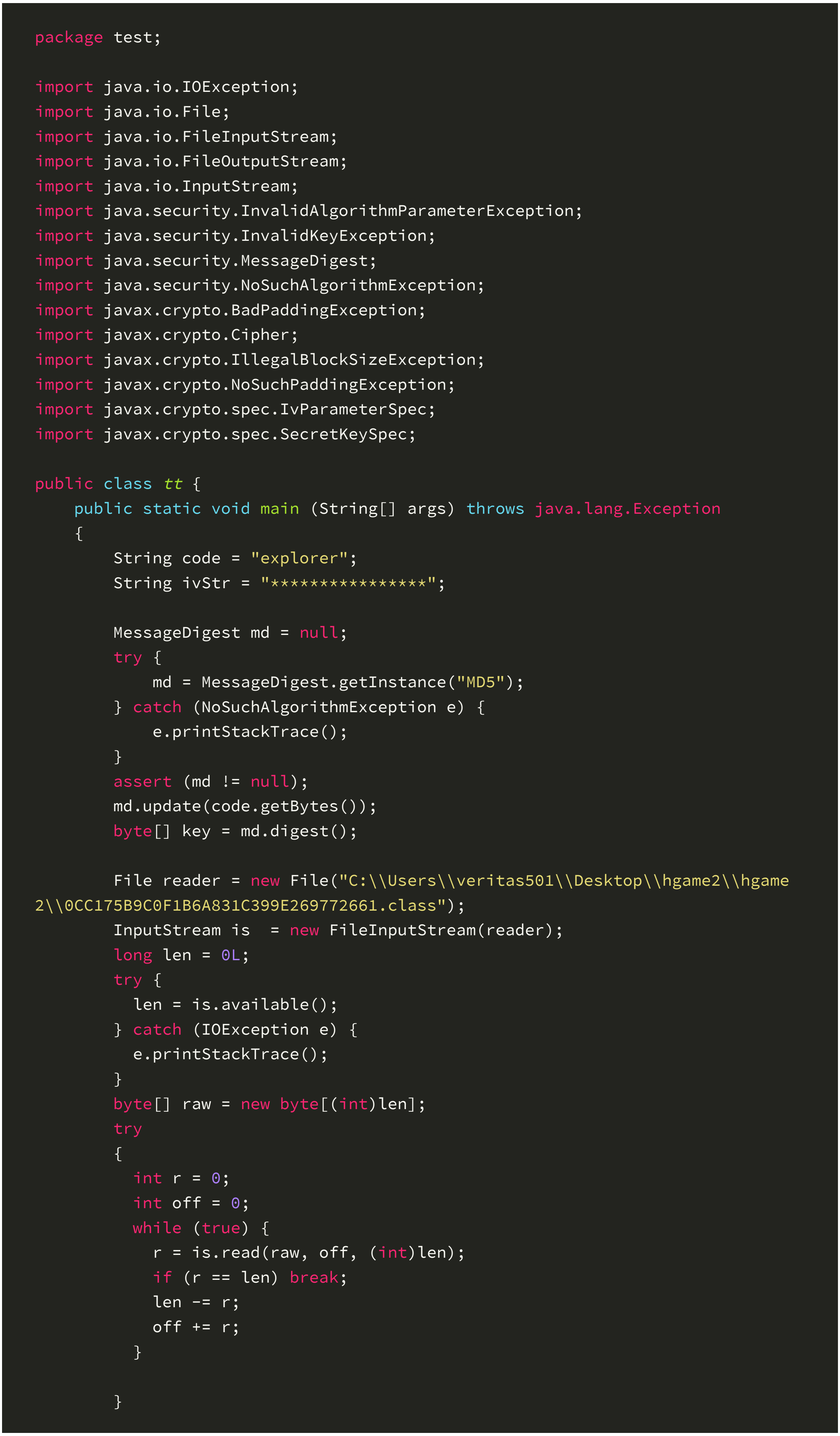
分析myClass知，class中的内容使用AES加密的。

key的生成：

```
private static String code = "explorer";
...
MessageDigest md = null;
try {
    md = MessageDigest.getInstance("MD5");
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
assert (md != null);
md.update(code.getBytes());
byte[] key = md.digest();
```

iv：`String ivStr = "****************";`

我们稍微修改一下代码，即可解密两个class：



```java
package test;

import java.io.IOException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class tt {
    public static void main (String[] args) throws java.lang.Exception
    {
        String code = "explorer";
        String ivStr = "****************";

        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        assert (md != null);
        md.update(code.getBytes());
        byte[] key = md.digest();

        File reader = new File("C:\\Users\\veritas501\\Desktop\\hgame2\\hgame2\\0CC175B9C0F1B6A831C399E269772661.class");
        InputStream is  = new FileInputStream(reader);
        long len = 0L;
        try {
          len = is.available();
        } catch (IOException e) {
          e.printStackTrace();
        }
        byte[] raw = new byte[(int)len];
        try
        {
          int r = 0;
          int off = 0;
          while (true) {
            r = is.read(raw, off, (int)len);
            if (r == len) break;
            len -= r;
            off += r;
          }

        }
        catch (IOException e)
        {
          e.printStackTrace();
        }

        Cipher cipher = null;
        try {
          cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        } catch (NoSuchAlgorithmException|NoSuchPaddingException e) {
          e.printStackTrace();
        }
        SecretKeySpec skey = new SecretKeySpec(key, "AES");
        IvParameterSpec iv = new IvParameterSpec(ivStr.getBytes());
        try {
          assert (cipher != null);
          cipher.init(2, skey, iv);
        } catch (InvalidKeyException|InvalidAlgorithmParameterException e) {
          e.printStackTrace();
        }

        byte[] en = null;
        try {
          en = cipher.doFinal(raw);
        } catch (IllegalBlockSizeException|BadPaddingException e) {
          e.printStackTrace();
        }
        File f=new File("C:\\Users\\veritas501\\Desktop\\hgame2\\hgame2\\tes
t.class");
        if (f.exists()==false) {
            f.createNewFile();//create file if not exist
        }
        FileOutputStream fos=new FileOutputStream(f);
        fos.write(en);
        fos.flush();
        fos.close();
    }
}
```

解密后用压缩软件在塞到jar里，再次用jd-gui打开，得到class的代码：

a.class：

```java
package hgame2;

import java.util.Arrays;

public class a
{
  public static boolean check(String flag)
  {
    byte[] var100 = flag.getBytes();

    byte[] var72 = new byte[var100.length + 2];
    System.arraycopy(var100, 0, var72, 0, var100.length);
    byte[] var140 = new byte[var72.length / 3 * 4];
    byte[] var82 = var140;
    int var17 = 0;
    int var18 = 0;

    while (var17 < var100.length) {
      var82[var18] = (byte)(var72[var17] >>> 2 & 0x3F);
      var82[(var18 + 1)] = (byte)(var72[(var17 + 1)] >>> 4 & 0xF | var72[var17] << 4 & 0x3F);
      var82[(var18 + 2)] = (byte)(var72[(var17 + 2)] >>> 6 & 0x3 | var72[(var17 + 1)] << 2 & 0x3F);
      var82[(var18 + 3)] = (byte)(var72[(var17 + 2)] & 0x3F);
      var17 += 3;
      var18 += 4;
    }
    var17 = 0;

    while (var17 < var82.length) {
      int var10000 = var82[var17];

      if (var10000 < 26) {
        var82[var17] = (byte)(var82[var17] + 65);
      }
      else {
        var10000 = var82[var17];
        byte var10001 = 52;

        if (var10000 < var10001) {
          var82[var17] = (byte)(var82[var17] + 97 - 26);
        }
        else {
          var10000 = var82[var17];
          var10001 = 62;

          if (var10000 < var10001) {
            var82[var17] = (byte)(var82[var17] + 48 - 52);
          }
          else {
            var10000 = var82[var17];
            var10001 = 63;

            if (var10000 < var10001) {
              var82[var17] = 43;
            }
            else {
              var140 = var82;
              int var136 = var17;
              var140[var136] = 47;
            }
          }
        }
      }
      var17++;
    }

    int var10000 = var82.length;
    byte var10001 = 1;
    var17 = var10000 - var10001;

    while (var17 > var100.length * 4 / 3) {
      var82[var17] = 61;
      var17--;
    }

    for (int i = 0; i < var82.length; i++) {
      var82[i] = (byte)(var82[i] ^ 0xCC);
    }

    byte[] f = { -107, -74, -118, -92, -81, -1, -126, -127, -127, -117, -118,
-89, -106, -108, -122, -86, -127, -102, -126, -86, -127, -101, -7, -4, -106,
-108, -123, -74, -81, -1, -98, -122, -82, -95, -81, -15 };
    return Arrays.equals(var82, f);
  }
}
```

checkFlag.class：

```
package hgame2;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;

public class checkFlag
{
  public static void trueMain()
    throws IOException
  {
    System.out.print("Now give me flag: ");
    BufferedReader strin = new BufferedReader(new InputStreamReader(System.i
n));
    String flag = strin.readLine();
    if (a.check(flag))
      System.out.println("hctf{" + flag + "}");
    else
      System.out.println("try again");
  }
}
```

a.class是先对flag做base64编码，然后每一位和0xcc异或，python写出反代码：

```
import base64
c=[-107, -74, -118, -92, -81, -1, -126, -127, -127, -117, -118, -89, -106, -10
8, -122, -86, -127, -102, -126, -86, -127, -101, -7, -4, -106, -108, -123, -7
4, -81, -1, -98, -122, -82, -95, -81, -15 ]

for i in range(len(c)):
    if c[i]<0:
        c[i]+=256
    c[i] ^= 0xCC
    c[i] = chr(c[i])
c = ''.join(c)

print 'hctf{'+base64.b64decode(c)+'}'
```

得到flag：`hctf{c1assL0ader_1S_1nter3stIng}`

---

explorer的奇怪番外7

工具用的jeb。安卓逆向，首先想到的是找找有没有字符串做参考，来到`values\strings.xml`，发现字段：

```
<string name="check_flag">
        check_flag</string>
<string name="enter_password">
        enter password</string>
```

挺好的，再来到`values\public.xml`，找到字段：

```
<public id="0x7f04001a" name="check_flag" type="layout" />
...
<public id="0x7f060015" name="check_flag" type="string" />
...
<public id="0x7f0c0054" name="check_flag" type="id" />
```

最后通过type为“id”的public id (2131492948)找到了相关的代码：

```
protected void onCreate(Bundle arg3) {
        super.onCreate(arg3);
        this.setContentView(2130968602);
        this.editText = this.findViewById(2131492949);
        this.button = this.findViewById(2131492948);
        this.textView = this.findViewById(2131492950);
        this.button.setOnClickListener(new View$OnClickListener() {
            public void onClick(View arg11) {
                String v6 = checkFlag.this.editText.getText().toString();
                try {
                    MessageDigest v4 = MessageDigest.getInstance("MD5");
                    v4.update(v6.getBytes());
                    if(!Arrays.equals(v4.digest(), new byte[]{-73, 14, 42, 13, -123, 91, 77, -57, -79, -22, 52, -88, -87, -47, 3, 5})) {
                        return;
                    }

                    MessageDigest v7 = MessageDigest.getInstance("sha-256");
                    v7.update(v6.getBytes());
                    checkFlag.this.textView.setText("hctf{" + checkFlag.bytes2Hex(v7.digest()) + "}");
                }
                catch(NoSuchAlgorithmException v0) {
                    v0.printStackTrace();
                }
            }
        });
    }
```

代码的意思就是获取输入，对输入进行md5加密，加密结果和已知数据比较，如果想等，则对输入进行sha256加密，结果加上‘hctf{}’就是flag了。
得到flag_md5: `{-73, 14, 42, 13, -123, 91, 77, -57, -79, -22, 52, -88, -87, -47, 3, 5}`
把signed转换成unsigned再hex：

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

int main(void)
{
    unsigned char b[] = { -73, 14, 42, 13, -123, 91, 77, -57, -79, -22, 52, -88, -87, -47, 3, 5 };
    for (int i = 0; i < 16; i++)
    {
        printf("%.2x", b[i]);
    }
    system("pause");
    return 0;
}//b70e2a0d855b4dc7b1ea34a8a9d10305
```

得到flag_md5: `b70e2a0d855b4dc7b1ea34a8a9d10305`

md5在线解密：http://www.md5online.org/

结果为： `Gabriel`

sha256加密得：`0c030df5a4e7477d218012c0121ebce6d61bb8dc46e0a6c4f8e1cc8091b946a5`

最后flag：`hctf{0c030df5a4e7477d218012c0121ebce6d61bb8dc46e0a6c4f8e1cc8091b946a5}`

---

coder

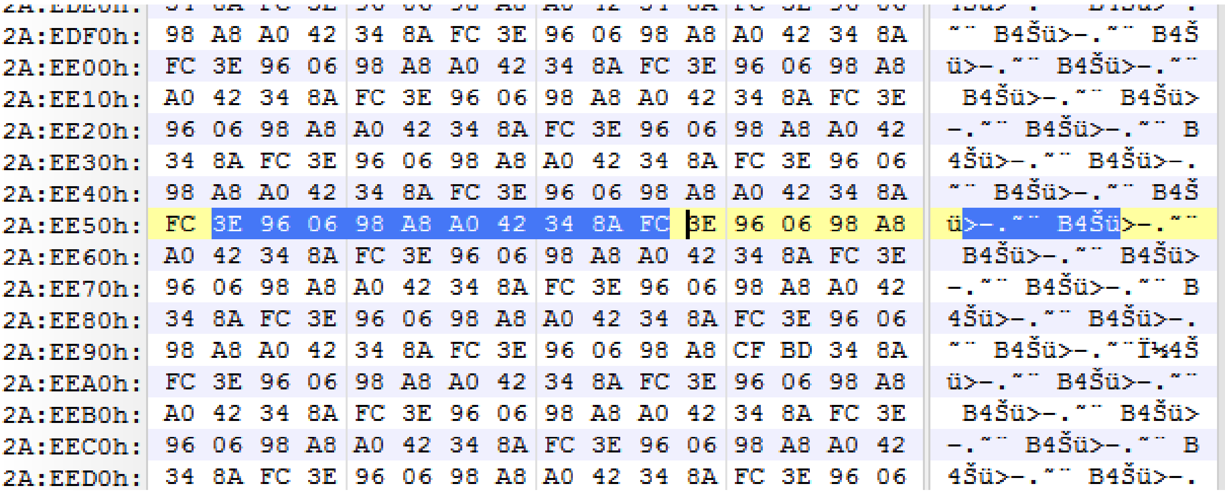
一开始，我是打算用正常的做re题的方法来解的，先跑跑，逆代码，分析，写反函数之类的来解，但我失败了，大概是水平不够。但我们依然有方法解题。

惯例扔到ida里，在main函数的加密函数中有如下一段：

```
    printf("encrypt ok, your key is ", buf);
    for ( k = 0; k <= 4; ++k )
      printf("%02x", *((_BYTE *)&pt_key1 + k));
    for ( l = 0; l <= 9; ++l )
      printf("%02x", *((_BYTE *)&pt_key2 + l));
    for ( m = 0; m < size; ++m )
    {
      *((_BYTE *)&pt_key1 + m % 5) = sub_401766(*((_BYTE *)&pt_key1 + m % 5), 2u);
      *((_BYTE *)&pt_key2 + m % 10) = sub_401766(*((_BYTE *)&pt_key2 + m % 10), 4u);
      *((_BYTE *)buf + m) ^= *((_BYTE *)&pt_key1 + m % 5) ^ *((_BYTE *)&pt_key2 + m % 10);// 将原始文件读入内存，在内存中加密，然后写出
    }
    v8 = open(*(const char **)(v12 + 24), 0x41, 0x1B6LL);
    write(v8, buf, size);
    close(v8);
    putchar(10);
    free(buf);
```

由此我们知道，key1为5位，key2为10位，加密过程可以大致表示为：`enc[i] ^= key3[i%10]`，其中key3为10位，`key3[i] = key1[i%5] ^ key2[i]`。

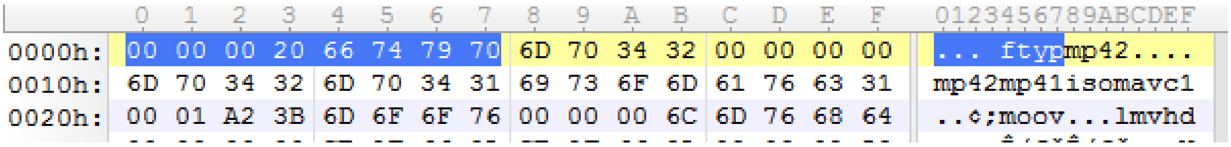
但我们现在不知道sub_401766函数是做什么的，起初我怀疑这个函数会对key做某种变换，导致每一轮加密用的key都不一样（满足某种函数关系），我试着分析了一下，但关系实在不好找，以为要GG，但我们打开flag.mp4文件，看到文件尾部为：



```
2A:EDF0h: 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 A0 42 34 8A  ˜¨ B4Šü>–.˜¨ B4Š
2A:EE00h: FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8  ü>–.˜¨ B4Šü>–.˜¨
2A:EE10h: A0 42 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E   B4Šü>–.˜¨ B4Šü>
2A:EE20h: 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 A0 42  –.˜¨ B4Šü>–.˜¨ B
2A:EE30h: 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06  4Šü>–.˜¨ B4Šü>–.
2A:EE40h: 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 A0 42 34 8A  ˜¨ B4Šü>–.˜¨ B4Š
2A:EE50h: FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8  ü>–.˜¨ B4Šü>–.˜¨
2A:EE60h: A0 42 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E   B4Šü>–.˜¨ B4Šü>
2A:EE70h: 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 A0 42  –.˜¨ B4Šü>–.˜¨ B
2A:EE80h: 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06  4Šü>–.˜¨ B4Šü>–.
2A:EE90h: 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 CF BD 34 8A  ˜¨ B4Šü>–.˜¨Ï½4Š
2A:EEA0h: FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8  ü>–.˜¨ B4Šü>–.˜¨
2A:EEB0h: A0 42 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E   B4Šü>–.˜¨ B4Šü>
2A:EEC0h: 96 06 98 A8 A0 42 34 8A FC 3E 96 06 98 A8 A0 42  –.˜¨ B4Šü>–.˜¨ B
2A:EED0h: 34 8A FC 3E 96 06 98 A8 A0 42 34 8A FC 3E 96 06  4Šü>–.˜¨ B4Šü>–.
```

非常整齐，十个一组，所以我的顾虑打消了，key3应该是不会改变的。

flag的格式为mp4，这个条件我们不能漏下，我随便打开了我硬盘上的几个mp4文件，发现前八个字节都是相同的，为 `dec1 = [0x00,0x00,0x00,0x20,0x66,0x74,0x79,0x70]`：

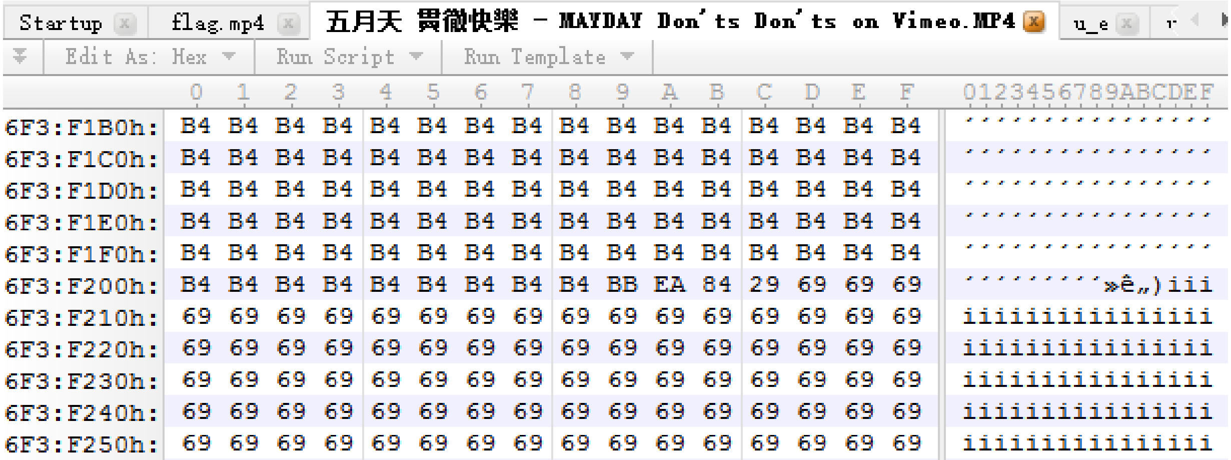


```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: 00 00 00 20 66 74 79 70 6D 70 34 32 00 00 00 00  ... ftypmp42....
0010h: 6D 70 34 32 6D 70 34 31 69 73 6F 6D 61 76 63 31  mp42mp41isomavc1
0020h: 00 01 A2 3B 6D 6F 6F 76 00 00 00 6C 6D 76 68 64  ..¢;moov...lmvhd
```

而加密后的文件头8个字节为：`enc1 = [0xFC,0x3E,0x96,0x1E,0xFE,0xDC,0xD9,0x32]`

我们将两者进行异或，得到 `xor1 = [0xFC,0x3E,0x96,0x3E,0x98,0xA8,0xA0,0x42]`

如果没有猜错，文件的结尾出应该是一串相同的字符，就像我手头的这个文件一样：



```
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
6F3:F1B0h: B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4  ´´´´´´´´´´´´´´´´
6F3:F1C0h: B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4  ´´´´´´´´´´´´´´´´
6F3:F1D0h: B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4  ´´´´´´´´´´´´´´´´
6F3:F1E0h: B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4  ´´´´´´´´´´´´´´´´
6F3:F1F0h: B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4  ´´´´´´´´´´´´´´´´
6F3:F200h: B4 B4 B4 B4 B4 B4 B4 B4 B4 BB EA 84 29 69 69 69  ´´´´´´´´´»ê„)iii
6F3:F210h: 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69  iiiiiiiiiiiiiiii
6F3:F220h: 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69  iiiiiiiiiiiiiiii
6F3:F230h: 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69  iiiiiiiiiiiiiiii
6F3:F240h: 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69  iiiiiiiiiiiiiiii
6F3:F250h: 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69  iiiiiiiiiiiiiiii
```

那么我们用python脚本：

```python
file_end = [0x3e,0x96,0x06,0x98,0xa8,0xa0,0x42,0x34,0x8a,0xfc,0x3e,0x96,0x06,0x98,0xa8,0xa0,0x42,0x34,0x8a,0xfc]
xor1 = [0xFC,0x3E,0x96,0x3E,0x98,0xA8,0xA0,0x42]

for i in range(10):
    print 'i=',i
    for j in range(8):
        print xor1[j]^file_end[i+j]
    print '===='
```

观察输出有如下一段：

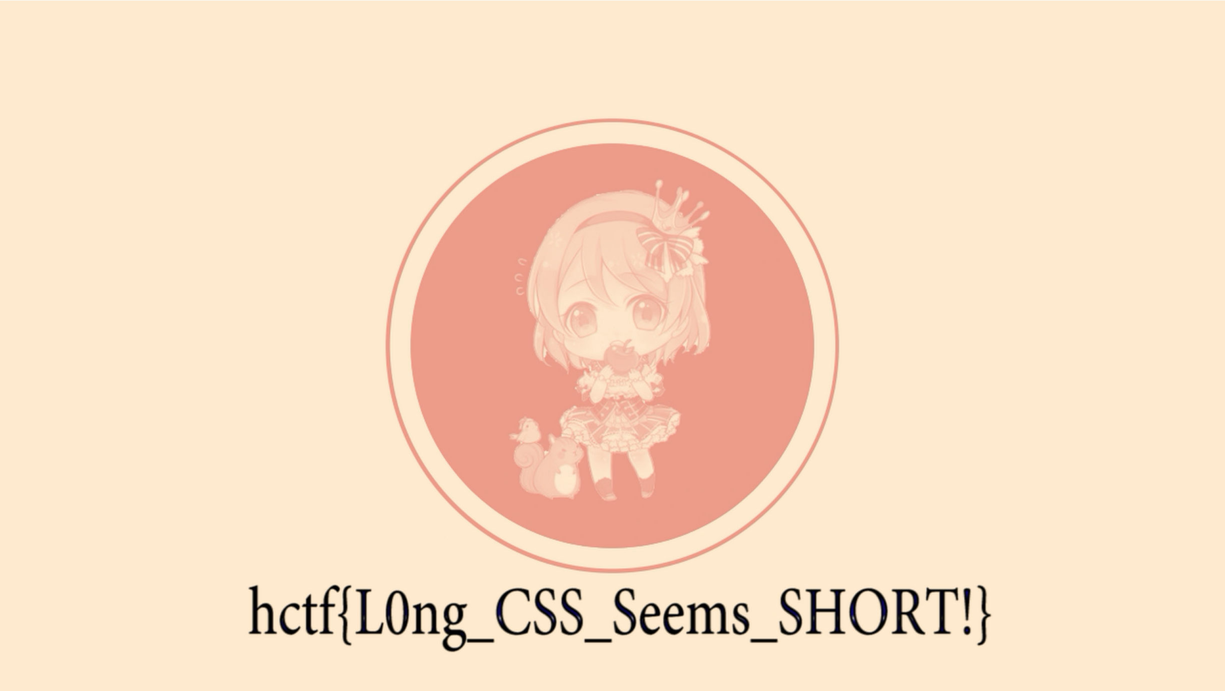
```
====
i= 9
0
0
0
56
0
0
0
0
====
```

这就验证了我们的猜想，只是flag.mp4的文件头的第4字节和我手头的MP4文件不同。根据file_end，从而我们得到了真正的 `key3 = [0xFC,0x3E,0x96,0x06,0x98,0xA8,0xA0,0x42,0x34,0x8a]`

我们利用key3对flag.mp4进行解密：

```
key3 = [0xFC,0x3E,0x96,0x06,0x98,0xA8,0xA0,0x42,0x34,0x8a]
i=0
fp = open(r'D:\flag.mp4','rb')
stream = list(fp.read())
fp.close()
out = []
for ch in stream:
    out.append(chr(ord(ch)^key3[i%10]))
    i += 1
out = ''.join(out)
fp = open(r'D:\flag_dec.mp4','wb')
fp.write(out)
fp.close()
```

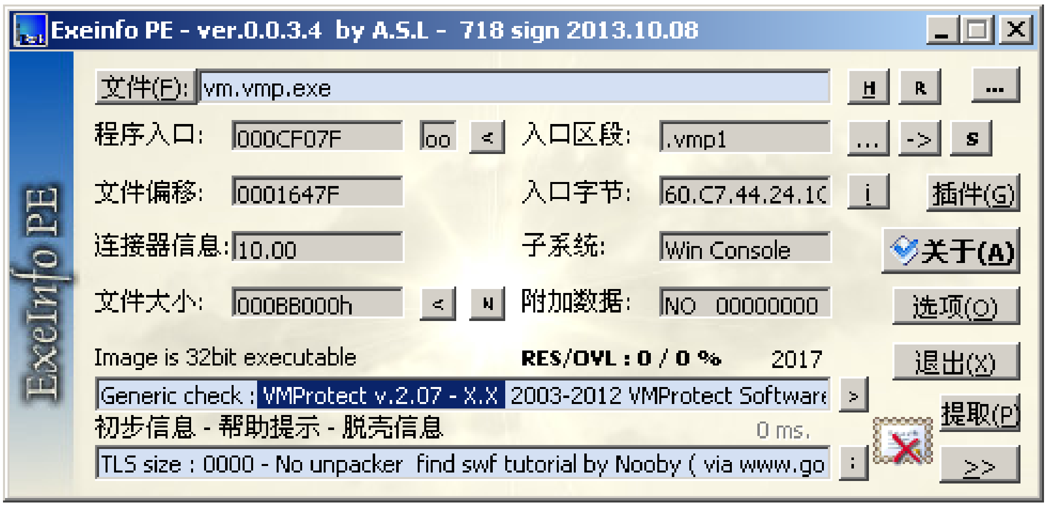
打开解密后的视频：



嗯，flag：`hctf{L0ng_CSS_Seems_SHORT!}`

easy-shell

首先扫下壳：

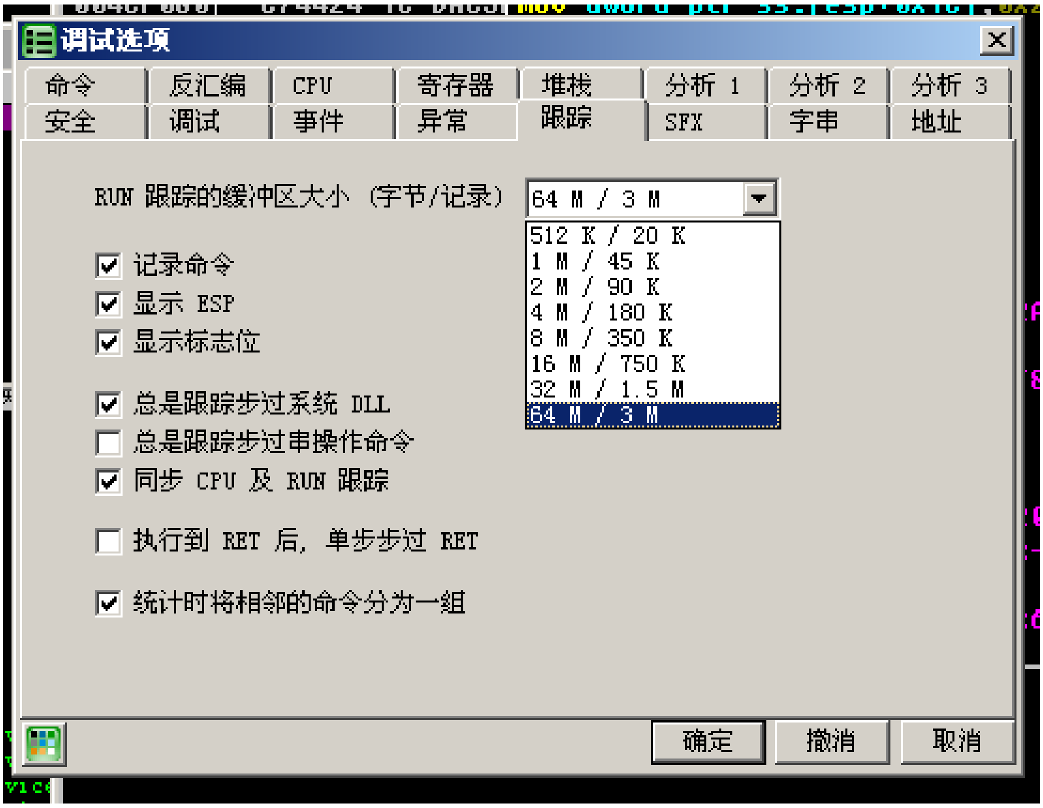


文件(F): vm.vmp.exe
程序入口: 000CF07F
入口区段: .vmp1
文件偏移: 0001647F
入口字节: 60.C7.44.24.1C
连接器信息: 10.00
子系统: Win Console
文件大小: 000BB000h
附加数据: NO 00000000
Image is 32bit executable RES/OVL : 0 / 0 % 2017
Generic check : VMProtect v.2.07 - X.X 2003-2012 VMProtect Software
初步信息 - 帮助提示 - 脱壳信息 0 ms.
TLS size : 0000 - No unpacker find swf tutorial by Nooby (via www.go

正如文件的名字，是个vmp壳，说真的我挺害怕这个壳的。

前排提醒：此处调试用的OD不能是原版OD，你可以使用网上各大论坛改的OD，比如52pojie，学破解，飘云阁等等。调试时需要Strong OD，FKVMP，忽略异常等。（你应该需要在XP上调试）

OD载入，先如下设置：



RUN 跟踪的缓冲区大小 (字节/记录) 64 M / 3 M

512 K / 20 K
1 M / 45 K
2 M / 90 K
4 M / 180 K
8 M / 350 K
16 M / 750 K
32 M / 1.5 M
64 M / 3 M

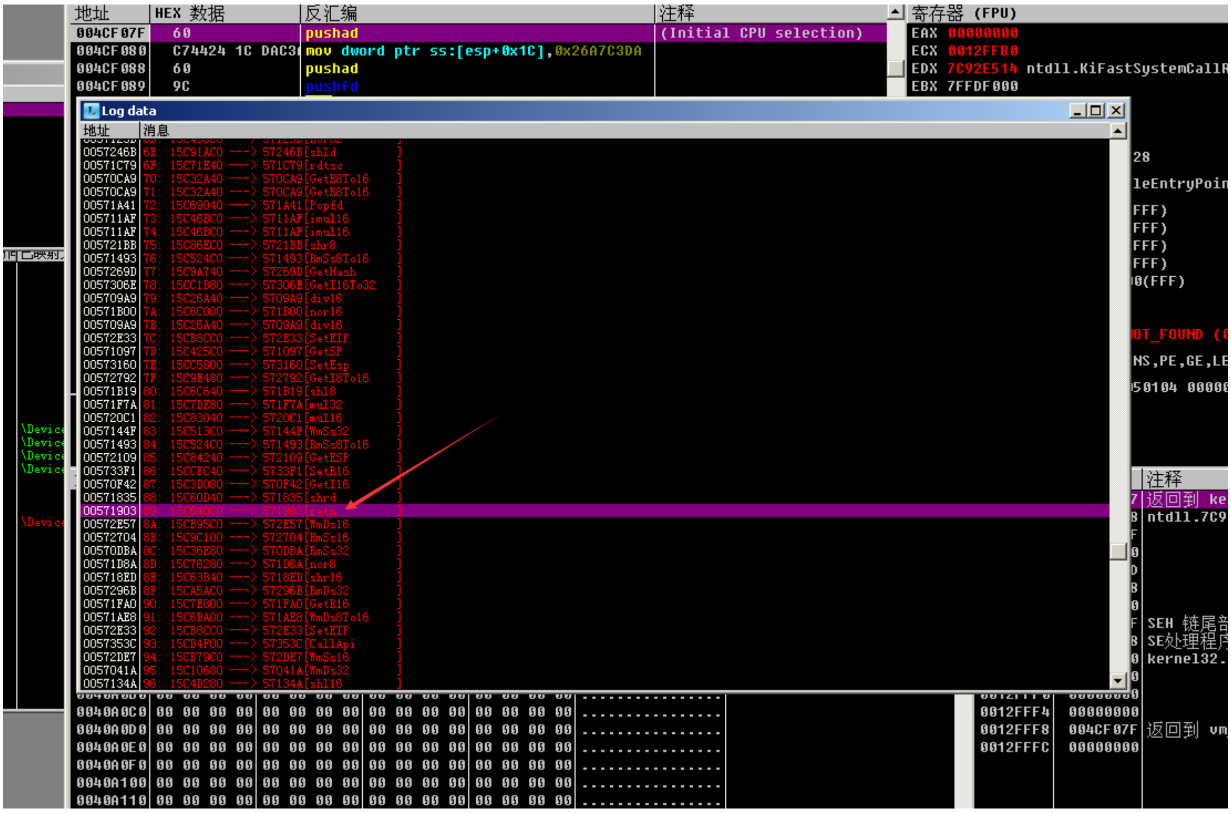
☑ 记录命令
☑ 显示 ESP
☑ 显示标志位

☑ 总是跟踪步过系统 DLL
□ 总是跟踪步过串操作命令
☑ 同步 CPU 及 RUN 跟踪

□ 执行到 RET 后，单步步过 RET

☑ 统计时将相邻的命令分为一组

在此时的代码处右键选择FKVMP>>start，点击OD上方的L按钮，找到retn：



```
004CF07F  60                pushad                                      (Initial CPU selection)
004CF080  C74424 1C DAC3(   mov dword ptr ss:[esp+0x1C],0x26A7C3DA
004CF088  60                pushad
004CF089  9C                pushfd
```

```
EAX 00000000
ECX 0012FFB0
EDX 7C92E514 ntdll.KiFastSystemCallR
EBX 7FFDF000
```

Log data

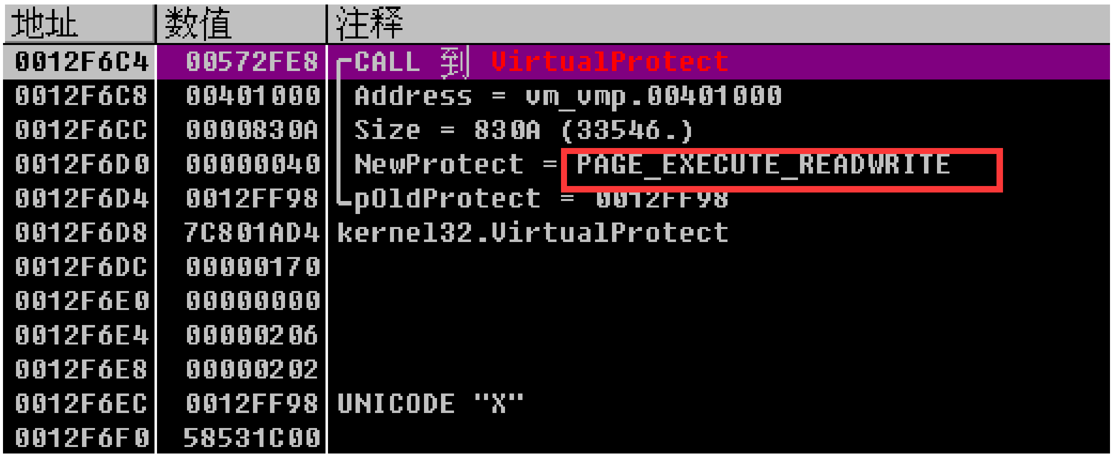
```
地址      消息
0057246B  6E: 15C91AC0 ---> 57246B[shld         ]
00571C79  6F: 15C71E40 ---> 571C79[rdtsc        ]
00570CA9  70: 15C32A40 ---> 570CA9[GetR8To16    ]
00570CA9  71: 15C32A40 ---> 570CA9[GetR8To16    ]
00571A41  72: 15C69040 ---> 571A41[Popfd        ]
005711AF  73: 15C46BC0 ---> 5711AF[imul16       ]
005711AF  74: 15C46BC0 ---> 5711AF[imul16       ]
005721BB  75: 15C86EC0 ---> 5721BB[shr8         ]
00571493  76: 15C524C0 ---> 571493[RmSs8To16    ]
0057269D  77: 15C9A740 ---> 57269D[GetHash      ]
0057306E  78: 15CC1B80 ---> 57306E[GetI16To32   ]
005709A9  79: 15C26A40 ---> 5709A9[div16        ]
00571B00  7A: 15C6C000 ---> 571B00[nor16        ]
005709A9  7B: 15C26A40 ---> 5709A9[div16        ]
00572E33  7C: 15CB8CC0 ---> 572E33[SetEIP       ]
00571097  7D: 15C425C0 ---> 571097[GetSP        ]
00573160  7E: 15CC5800 ---> 573160[SetEsp       ]
00572792  7F: 15C9E480 ---> 572792[GetI8To16    ]
00571B19  80: 15C6C640 ---> 571B19[shl8         ]
00571F7A  81: 15C7DE80 ---> 571F7A[mul32        ]
005720C1  82: 15C83040 ---> 5720C1[mul16        ]
0057144F  83: 15C513C0 ---> 57144F[WmSs32       ]
00571493  84: 15C524C0 ---> 571493[RmSs8To16    ]
00572109  85: 15C84240 ---> 572109[GetESP       ]
005733F1  86: 15CCFC40 ---> 5733F1[SetR16       ]
00570F42  87: 15C3D080 ---> 570F42[GetI16       ]
00571835  88: 15C60D40 ---> 571835[shrd         ]
00571903  89: 15C640C0 ---> 571903[retn         ]
00572E57  8A: 15CB95C0 ---> 572E57[WmDs16       ]
00572704  8B: 15C9C100 ---> 572704[RmSs16       ]
00570DBA  8C: 15C36E80 ---> 570DBA[RmSs32       ]
00571D8A  8D: 15C76280 ---> 571D8A[nor8         ]
005718ED  8E: 15C63B40 ---> 5718ED[shr16        ]
0057296B  8F: 15CA5AC0 ---> 57296B[RmDs32       ]
00571FA0  90: 15C7E800 ---> 571FA0[GetR16       ]
00571AE8  91: 15C6BA00 ---> 571AE8[WmDs8To16    ]
00572E33  92: 15CB8CC0 ---> 572E33[SetEIP       ]
0057353C  93: 15CD4F00 ---> 57353C[CallApi      ]
00572DE7  94: 15CB79C0 ---> 572DE7[WmSs16       ]
0057041A  95: 15C10680 ---> 57041A[WmDs32       ]
0057134A  96: 15C4D280 ---> 57134A[shl16        ]
```

```
0040A0C0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0040A0D0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0040A0E0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0040A0F0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0040A100  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0040A110  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

```
0012FFF4  00000000
0012FFF8  004CF07F  返回到 vm
0012FFFC  00000000
```

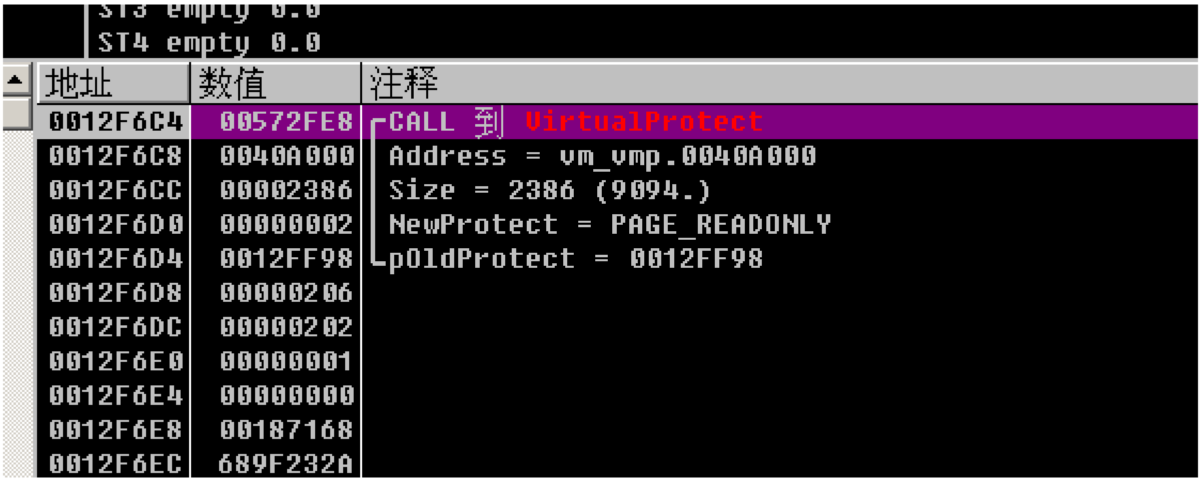
记录retn的地址：`0x00571903`

接着在OD上下断点：`bp VirtualProtect`，按f9运行，观察堆栈区：



```
地址      数值      注释
0012F6C4  00572FE8  ┌CALL 到 VirtualProtect
0012F6C8  00401000  │Address = vm_vmp.00401000
0012F6CC  0000830A  │Size = 830A (33546.)
0012F6D0  00000040  │NewProtect = PAGE_EXECUTE_READWRITE
0012F6D4  0012FF98  └pOldProtect = 0012FF98
0012F6D8  7C801AD4  kernel32.VirtualProtect
0012F6DC  00000170
0012F6E0  00000000
0012F6E4  00000206
0012F6E8  00000202
0012F6EC  0012FF98  UNICODE "X"
0012F6F0  58531C00
```

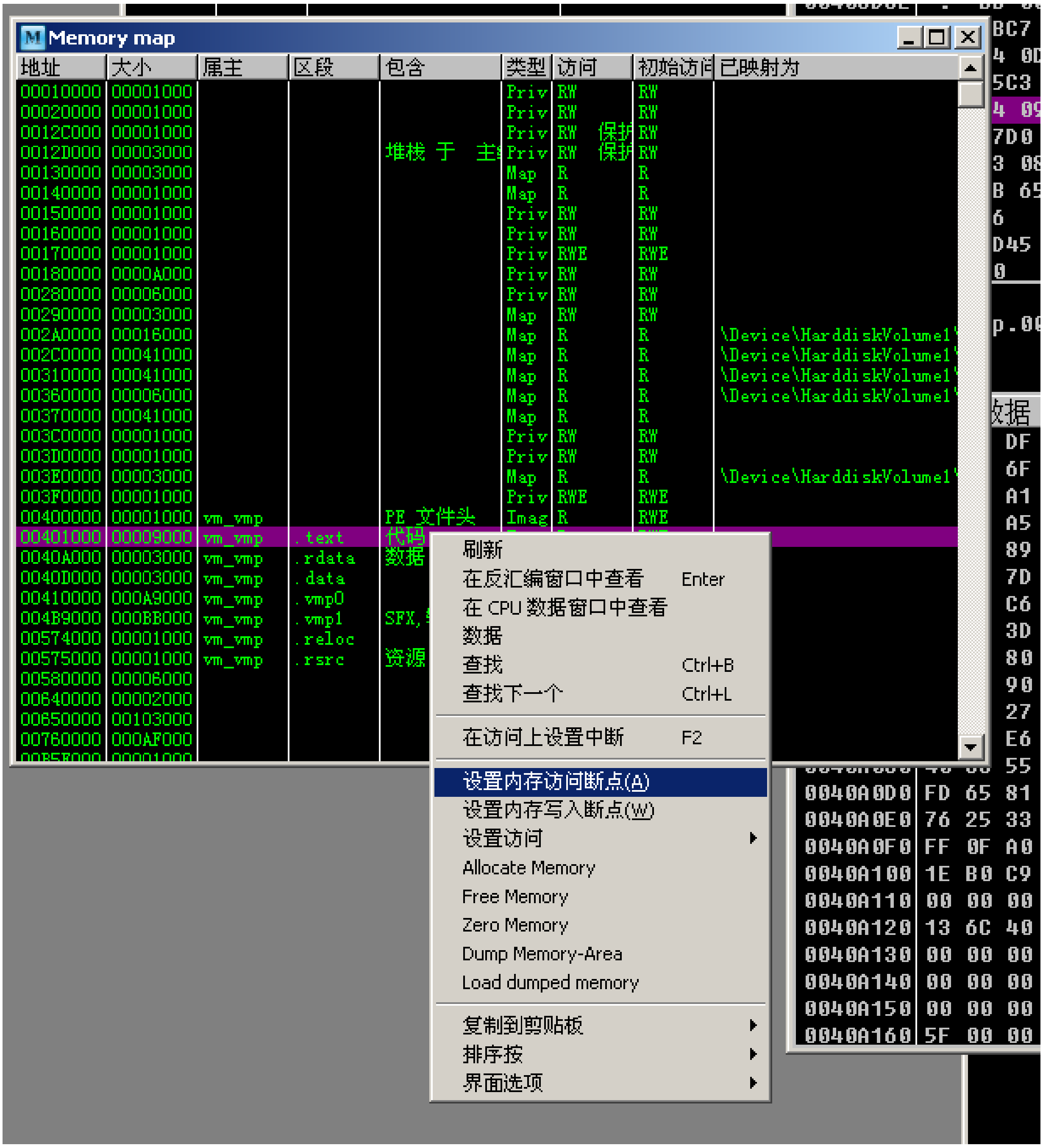
直到NewProtect = READONLY：



```
ST3 empty 0.0
ST4 empty 0.0
```

```
地址      数值      注释
0012F6C4  00572FE8  ┌CALL 到 VirtualProtect
0012F6C8  0040A000  │Address = vm_vmp.0040A000
0012F6CC  00002386  │Size = 2386 (9094.)
0012F6D0  00000002  │NewProtect = PAGE_READONLY
0012F6D4  0012FF98  └pOldProtect = 0012FF98
0012F6D8  00000206
0012F6DC  00000202
0012F6E0  00000001
0012F6E4  00000000
0012F6E8  00187168
0012F6EC  689F232A
```

此时alt+B，断点界面取消或禁用断点，然后alt+M，对text段下内存访问断点：

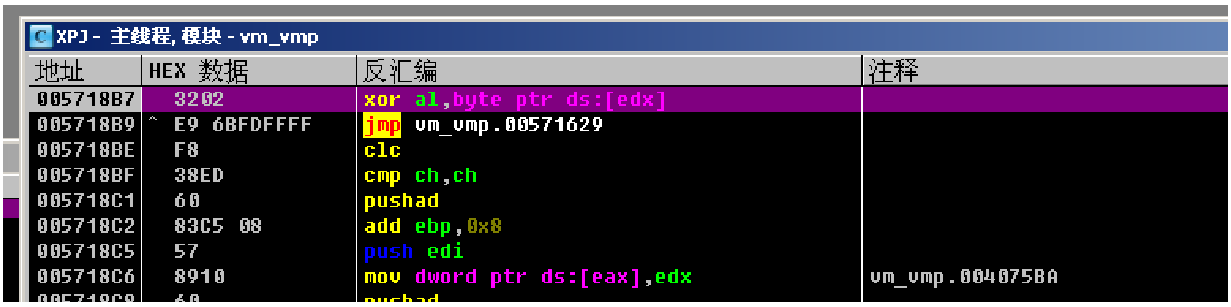


Memory map

地址	大小	属主	区段	包含	类型	访问	初始访问	已映射为
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0012C000	00001000				Priv	RW 保护	RW	
0012D000	00003000			堆栈 于 主	Priv	RW 保护	RW	
00130000	00003000				Map	R	R	
00140000	00001000				Map	R	R	
00150000	00001000				Priv	RW	RW	
00160000	00001000				Priv	RW	RW	
00170000	00001000				Priv	RWE	RWE	
00180000	0000A000				Priv	RW	RW	
00280000	00006000				Priv	RW	RW	
00290000	00003000				Map	RW	RW	
002A0000	00016000				Map	R	R	\Device\HarddiskVolume1\
002C0000	00041000				Map	R	R	\Device\HarddiskVolume1\
00310000	00041000				Map	R	R	\Device\HarddiskVolume1\
00360000	00006000				Map	R	R	\Device\HarddiskVolume1\
00370000	00041000				Map	R	R	
003C0000	00001000				Priv	RW	RW	
003D0000	00001000				Priv	RW	RW	
003E0000	00003000				Map	R	R	\Device\HarddiskVolume1\
003F0000	00001000				Priv	RWE	RWE	
00400000	00001000	vm_vmp		PE 文件头	Imag	R	RWE	
00401000	00009000	vm_vmp	.text	代码				
0040A000	00003000	vm_vmp	.rdata	数据				
0040D000	00003000	vm_vmp	.data					
00410000	000A9000	vm_vmp	.vmp0					
004B9000	000BB000	vm_vmp	.vmp1	SFX,				
00574000	00001000	vm_vmp	.reloc					
00575000	00001000	vm_vmp	.rsrc	资源				
00580000	00006000							
00640000	00002000							
00650000	00103000							
00760000	000AF000							

```
0040A0D0 FD 65 81
0040A0E0 76 25 33
0040A0F0 FF 0F A0
0040A100 1E B0 C9
0040A110 00 00 00
0040A120 13 6C 40
0040A130 00 00 00
0040A140 00 00 00
0040A150 00 00 00
0040A160 5F 00 00
```

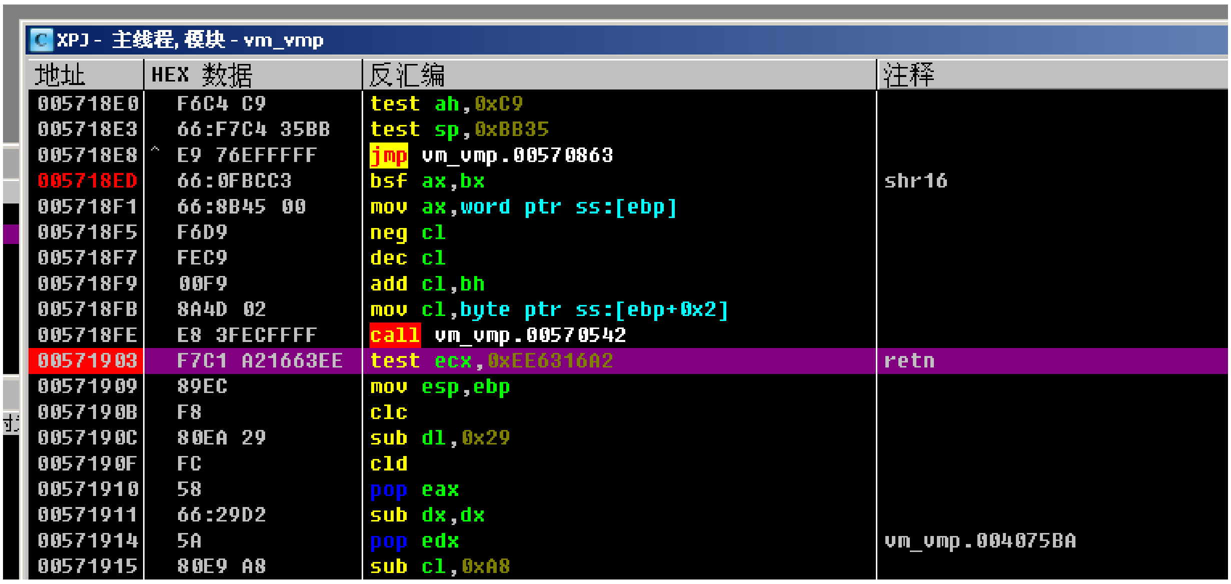
f9一下，取消text段的访问断点，来到这里：



XPJ - 主线程, 模块 - vm_vmp

```
地址      HEX 数据        反汇编                       注释
005718B7  3202           xor al,byte ptr ds:[edx]
005718B9 ^ E9 6BFDFFFF   jmp vm_vmp.00571629
005718BE  F8             clc
005718BF  38ED           cmp ch,ch
005718C1  60             pushad
005718C2  83C5 08        add ebp,0x8
005718C5  57             push edi
005718C6  8910           mov dword ptr ds:[eax],edx   vm_vmp.004075BA
```

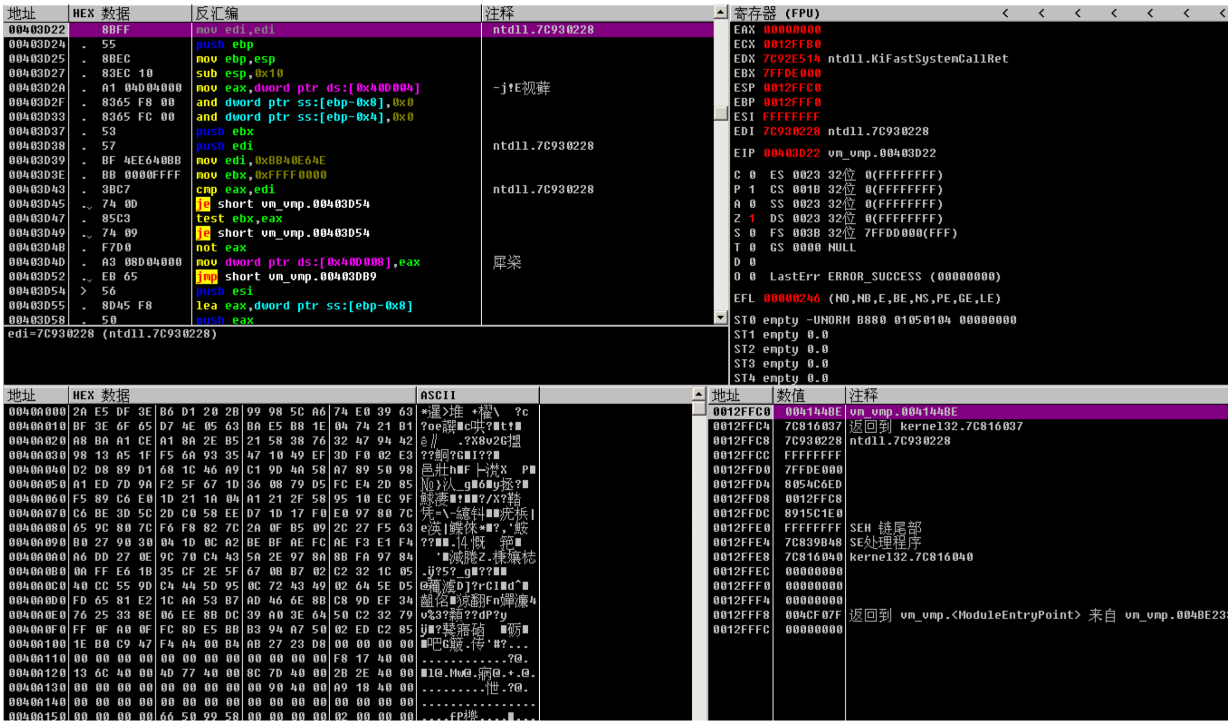
掏出我们之前记录下的retn地址：ctrl+G转到然后f2下断：



XPJ - 主线程, 模块 - vm_vmp

```
地址      HEX 数据         反汇编                          注释
005718E0   F6C4 C9         test ah,0xC9
005718E3   66:F7C4 35BB    test sp,0xBB35
005718E8 ^ E9 76EFFFFF     jmp vm_vmp.00570863
005718ED   66:0FBCC3       bsf ax,bx                       shr16
005718F1   66:8B45 00      mov ax,word ptr ss:[ebp]
005718F5   F6D9            neg cl
005718F7   FEC9            dec cl
005718F9   00F9            add cl,bh
005718FB   8A4D 02         mov cl,byte ptr ss:[ebp+0x2]
005718FE   E8 3FECFFFF     call vm_vmp.00570542
00571903   F7C1 A21663EE   test ecx,0xEE6316A2             retn
00571909   89EC            mov esp,ebp
0057190B   F8              clc
0057190C   80EA 29         sub dl,0x29
0057190F   FC              cld
00571910   58              pop eax
00571911   66:29D2         sub dx,dx
00571914   5A              pop edx                         vm_vmp.004075BA
00571915   80E9 A8         sub cl,0xA8
```

f9一下，取消retn的断点，再对text段下内存访问断点，f9一下，来到了我们的oep：



```
地址      HEX 数据          反汇编                               注释
00403D22    8BFF            mov edi,edi                          ntdll.7C930228
00403D24  . 55              push ebp
00403D25  . 8BEC            mov ebp,esp
00403D27  . 83EC 10         sub esp,0x10
00403D2A  . A1 04D04000     mov eax,dword ptr ds:[0x40D004]      -j!E视薛
00403D2F  . 8365 F8 00      and dword ptr ss:[ebp-0x8],0x0
00403D33  . 8365 FC 00      and dword ptr ss:[ebp-0x4],0x0
00403D37  . 53              push ebx
00403D38  . 57              push edi                             ntdll.7C930228
00403D39  . BF 4EE640BB     mov edi,0xBB40E64E
00403D3E  . BB 0000FFFF     mov ebx,0xFFFF0000
00403D43  . 3BC7            cmp eax,edi                          ntdll.7C930228
00403D45  .v 74 0D          je short vm_vmp.00403D54
00403D47  . 85C3            test ebx,eax
00403D49  .v 74 09          je short vm_vmp.00403D54
00403D4B  . F7D0            not eax
00403D4D  . A3 08D04000     mov dword ptr ds:[0x40D008],eax      犀粢
00403D52  .v EB 65          jmp short vm_vmp.00403DB9
00403D54  > 56              push esi
00403D55  . 8D45 F8         lea eax,dword ptr ss:[ebp-0x8]
00403D58  . 50              push eax
edi=7C930228 (ntdll.7C930228)
```

```
寄存器 (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C92E514 ntdll.KiFastSystemCallRet
EBX 7FFDE000
ESP 0012FFC0
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C930228 ntdll.7C930228

EIP 00403D22 vm_vmp.00403D22

C 0  ES 0023 32位 0(FFFFFFFF)
P 1  CS 001B 32位 0(FFFFFFFF)
A 0  SS 0023 32位 0(FFFFFFFF)
Z 1  DS 0023 32位 0(FFFFFFFF)
S 0  FS 003B 32位 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM B880 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
```

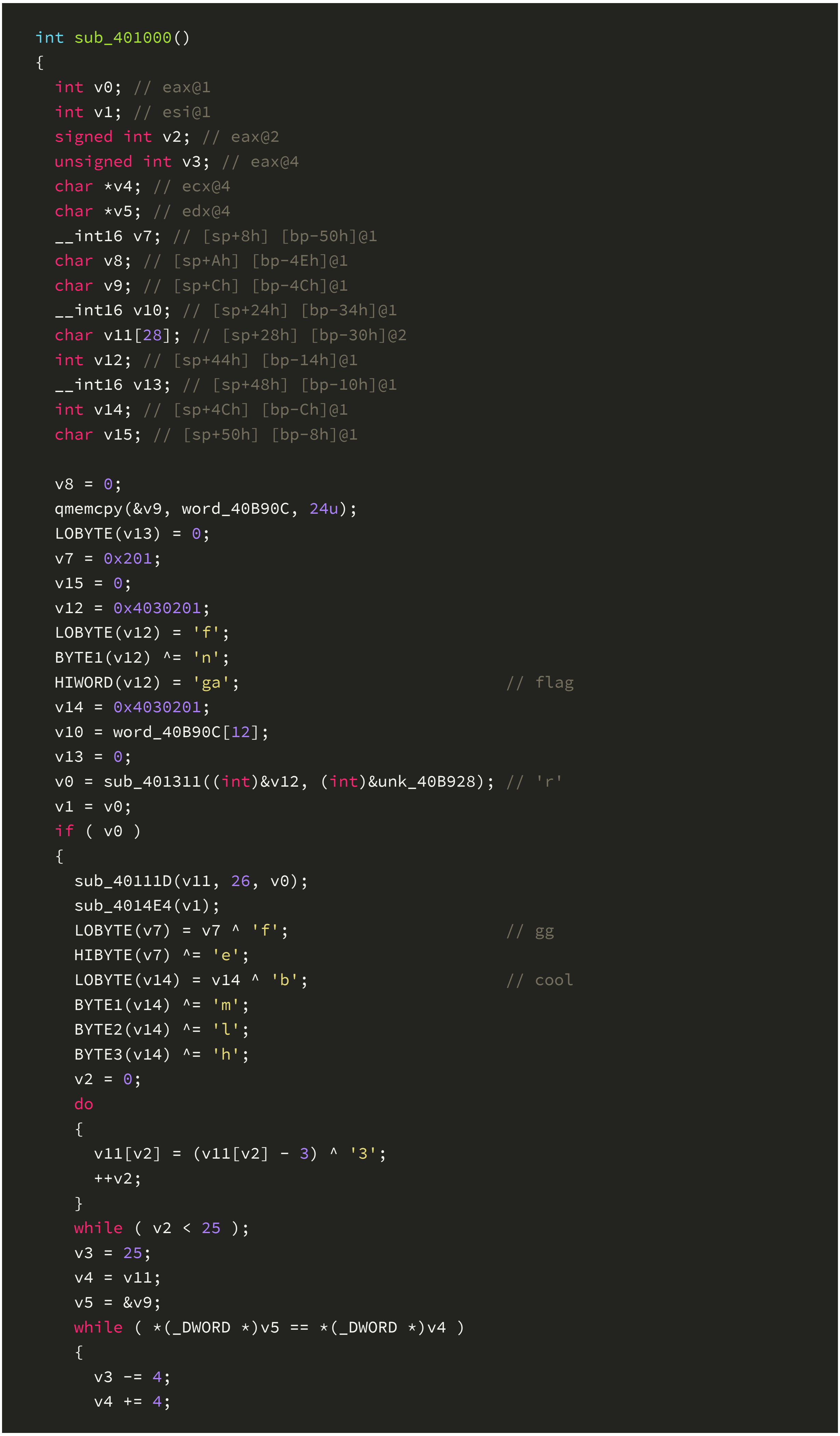
```
地址      HEX 数据                                         ASCII
0040A000  2A E5 DF 3E B6 D1 20 2B 99 98 5C A6 74 E0 39 63  *暹>堆 +櫂\ ?c
0040A010  BF 3E 6F 65 D7 4E 05 63 BA E5 B8 1E 04 74 21 B1  ?oe譔■c哄?■t!■
0040A020  A8 BA A1 CE A1 8A 2E B5 21 58 38 76 32 47 94 42  ǎ∥ .?X8v2G攌
0040A030  98 13 A5 1F F5 6A 93 35 47 10 49 EF 3D F0 02 E3  ??鮦?G■I??■
0040A040  D2 D8 89 D1 68 1C 46 A9 C1 9D 4A 58 A7 89 50 98  邑壯h■F├澂X P■
0040A050  A1 ED 7D 9A F2 5F 67 1D 36 08 79 D5 FC E4 2D 85  №}氿_g■6■y拯?■
0040A060  F5 89 C6 E0 1D 21 1A 04 A1 21 2F 58 95 10 EC 9F  鮇凄■!■■?/X?鞊
0040A070  C6 BE 3D 5C 2D C0 58 EE D7 1D 17 F0 E0 97 80 7C  凭=\-繶钭■■疺梹|
0040A080  65 9C 80 7C F6 F8 82 7C 2A 0F B5 09 2C 27 F5 63  e渶|鯠俅*■?,'鮟
0040A090  B0 27 90 30 04 1D 0C A2 BE BF AE FC AE F3 E1 F4  ??■■.14 慨 鈪■
0040A0A0  A6 DD 27 0E 9C 70 C4 43 5A 2E 97 8A 8B FA 97 84  '■減腃Z.棅嫽梽
0040A0B0  0A FF E6 1B 35 CF 2E 5F 67 0B B7 02 C2 32 1C 05  .ÿ?5?_g■??■■
0040A0C0  40 CC 55 9D C4 44 5D 95 0C 72 43 49 02 64 5E D5  @蘌漳D]?rCI■d^■
0040A0D0  FD 65 81 E2 1C AA 53 B7 AD 46 6E 8B C8 9D EF 34  齟佲■猄翻Fn嫿潄4
0040A0E0  76 25 33 8E 06 EE 8B DC 39 A0 3E 64 50 C2 32 79  v%3?顙??dP?y
0040A0F0  FF 0F A0 0F FC 8D E5 BB B3 94 A7 50 02 ED C2 85  ÿ■?鼜寤硵 ■砺■
0040A100  1E B0 C9 47 F4 A4 00 B4 AB 27 23 D8 00 00 00 00  ■吧G簸.传'#?...
0040A110  00 00 00 00 00 00 00 00 00 00 00 00 F8 17 40 00  ............?@.
0040A120  13 6C 40 00 4D 77 40 00 8C 7D 40 00 2B 2E 40 00  ■l@.Mw@.寎@.+.@.
0040A130  00 00 00 00 00 00 00 00 00 90 40 00 A9 18 40 00  .........恅.?@.
0040A140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040A150  00 00 00 00 66 50 99 58 00 00 00 00 02 00 00 00  ....fP橷....■...
```

```
地址      数值      注释
0012FFC0  004144BE  vm_vmp.004144BE
0012FFC4  7C816037  返回到 kernel32.7C816037
0012FFC8  7C930228  ntdll.7C930228
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
0012FFD4  8054C6ED
0012FFD8  0012FFC8
0012FFDC  8915C1E0
0012FFE0  FFFFFFFF  SEH 链尾部
0012FFE4  7C839B48  SE处理程序
0012FFE8  7C816040  kernel32.7C816040
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  004CF07F  返回到 vm_vmp.<ModuleEntryPoint> 来自 vm_vmp.004BE23
0012FFFC  00000000
```

（此时我们可以对oep下硬件执行断点，方便下次调试，不用再重复之前那些动作）

我们现在可以用lordpe dump一下镜像，虽然IAT没有修复，但是IDA还是能分析部分的。

ida载入，找到关键函数：

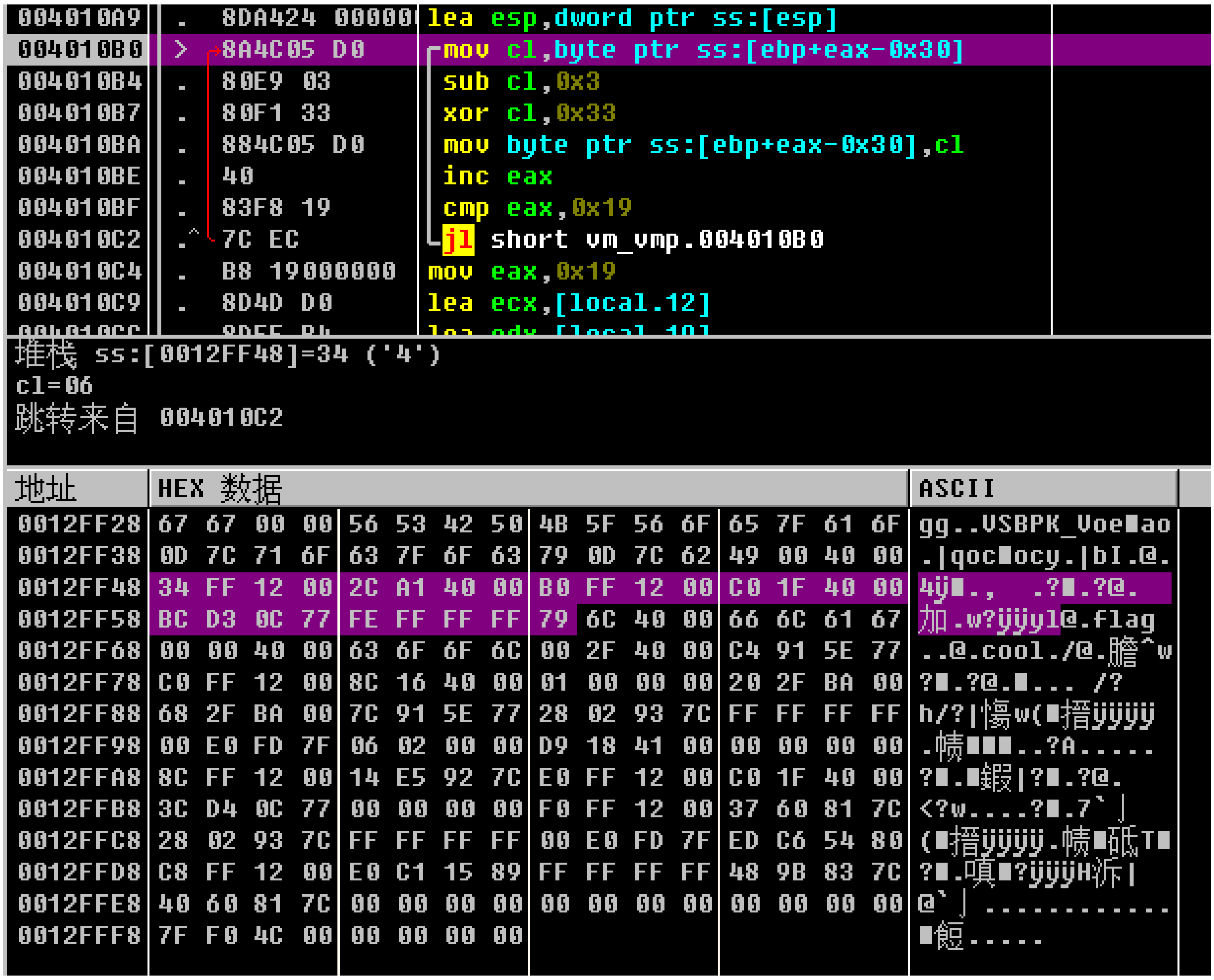


```
int sub_401000()
{
  int v0; // eax@1
  int v1; // esi@1
  signed int v2; // eax@2
  unsigned int v3; // eax@4
  char *v4; // ecx@4
  char *v5; // edx@4
  __int16 v7; // [sp+8h] [bp-50h]@1
  char v8; // [sp+Ah] [bp-4Eh]@1
  char v9; // [sp+Ch] [bp-4Ch]@1
  __int16 v10; // [sp+24h] [bp-34h]@1
  char v11[28]; // [sp+28h] [bp-30h]@2
  int v12; // [sp+44h] [bp-14h]@1
  __int16 v13; // [sp+48h] [bp-10h]@1
  int v14; // [sp+4Ch] [bp-Ch]@1
  char v15; // [sp+50h] [bp-8h]@1

  v8 = 0;
  qmemcpy(&v9, word_40B90C, 24u);
  LOBYTE(v13) = 0;
  v7 = 0x201;
  v15 = 0;
  v12 = 0x4030201;
  LOBYTE(v12) = 'f';
  BYTE1(v12) ^= 'n';
  HIWORD(v12) = 'ga';                           // flag
  v14 = 0x4030201;
  v10 = word_40B90C[12];
  v13 = 0;
  v0 = sub_401311((int)&v12, (int)&unk_40B928); // 'r'
  v1 = v0;
  if ( v0 )
  {
    sub_40111D(v11, 26, v0);
    sub_4014E4(v1);
    LOBYTE(v7) = v7 ^ 'f';                      // gg
    HIBYTE(v7) ^= 'e';
    LOBYTE(v14) = v14 ^ 'b';                    // cool
    BYTE1(v14) ^= 'm';
    BYTE2(v14) ^= 'l';
    BYTE3(v14) ^= 'h';
    v2 = 0;
    do
    {
      v11[v2] = (v11[v2] - 3) ^ '3';
      ++v2;
    }
    while ( v2 < 25 );
    v3 = 25;
    v4 = v11;
    v5 = &v9;
    while ( *(_DWORD *)v5 == *(_DWORD *)v4 )
    {
      v3 -= 4;
      v4 += 4;
      v5 += 4;
      if ( v3 < 4 )
      {
        if ( *v4 != *v5 )
          break;
        sub_401328(&v14);
        return 0;
      }
    }
  }
  sub_401328(&v7);
  return 0;
}
```

忽略中间的n多细节，我们只看两段：

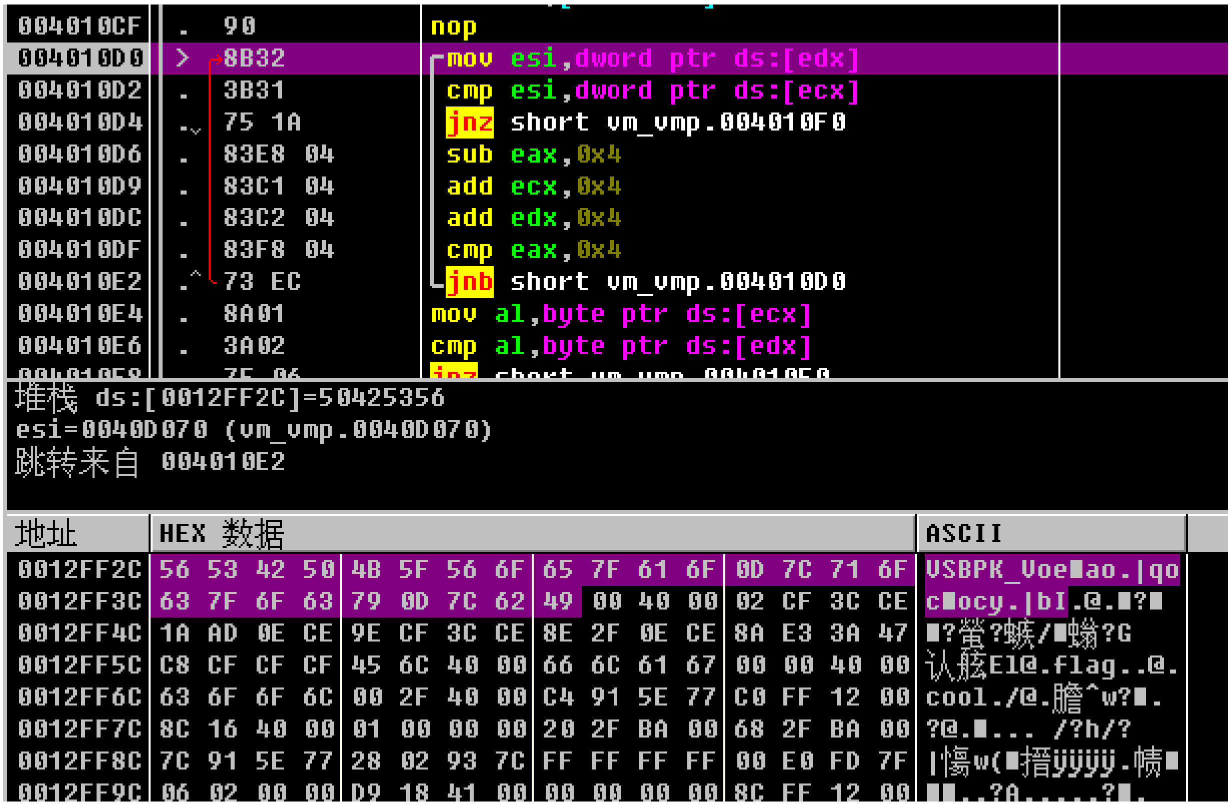
```
do
    {
      v11[v2] = (v11[v2] - 3) ^ '3';
      ++v2;
    }
while ( v2 < 25 );
...
while ( *(_DWORD *)v5 == *(_DWORD *)v4 )
{
    v3 -= 4;
    v4 += 4;
    v5 += 4;
    if ( v3 < 4 )
    {
        if ( *v4 != *v5 )
            break;
        sub_401328(&v14); //打印函数（cool）
        return 0;
    }
}
sub_401328(&v7);//打印函数（GG）
```

结合OD动态调试，我们发现，函数是将写在0x0012ff48处的数据做`byte = (byte - 3) ^ '3'`变换：



```
004010A9  .   8DA424 00000  lea esp,dword ptr ss:[esp]
004010B0  >  ┌8A4C05 D0     ┌mov cl,byte ptr ss:[ebp+eax-0x30]
004010B4  .  │80E9 03       │sub cl,0x3
004010B7  .  │80F1 33       │xor cl,0x33
004010BA  .  │884C05 D0     │mov byte ptr ss:[ebp+eax-0x30],cl
004010BE  .  │40            │inc eax
004010BF  .  │83F8 19       │cmp eax,0x19
004010C2  .^ └7C EC         └jl short vm_vmp.004010B0
004010C4  .   B8 19000000   mov eax,0x19
004010C9  .   8D4D D0       lea ecx,[local.12]
004010CC  .   8D55 D4       lea edx,[local.10]
堆栈 ss:[0012FF48]=34 ('4')
cl=06
跳转来自 004010C2

地址      HEX 数据                                          ASCII
0012FF28  67 67 00 00 56 53 42 50 4B 5F 56 6F 65 7F 61 6F  gg..VSBPK_Voe■ao
0012FF38  0D 7C 71 6F 63 7F 6F 63 79 0D 7C 62 49 00 40 00  .|qoc■ocy.|bI.@.
0012FF48  34 FF 12 00 2C A1 40 00 B0 FF 12 00 C0 1F 40 00  4ÿ■.,  .?■.?@.
0012FF58  BC D3 0C 77 FE FF FF FF 79 6C 40 00 66 6C 61 67  加.w?ÿÿyl@.flag
0012FF68  00 00 40 00 63 6F 6F 6C 00 2F 40 00 C4 91 5E 77  ..@.cool./@.牄^w
0012FF78  C0 FF 12 00 8C 16 40 00 01 00 00 00 20 2F BA 00  ?■.?@.■... /?
0012FF88  68 2F BA 00 7C 91 5E 77 28 02 93 7C FF FF FF FF  h/?|憓w(■搢ÿÿÿÿ
0012FF98  00 E0 FD 7F 06 02 00 00 D9 18 41 00 00 00 00 00  .帻■■■..?A.....
0012FFA8  8C FF 12 00 14 E5 92 7C E0 FF 12 00 C0 1F 40 00  ?■.■鍜|?■.?@.
0012FFB8  3C D4 0C 77 00 00 00 00 F0 FF 12 00 37 60 81 7C  <?w....?■.7`亅
0012FFC8  28 02 93 7C FF FF FF FF 00 E0 FD 7F ED C6 54 80  (■搢ÿÿÿÿ.帻■砥T■
0012FFD8  C8 FF 12 00 E0 C1 15 89 FF FF FF FF 48 9B 83 7C  ?■.嗔■?ÿÿÿH泝|
0012FFE8  40 60 81 7C 00 00 00 00 00 00 00 00 00 00 00 00  @`亅............
0012FFF8  7F F0 4C 00 00 00 00 00                          ■餖.....
```

然后和写在0x0012FF2C处的数据一位一位比较，如果相同则输出’cool‘：



```
004010CF  .   90            nop
004010D0  >  ┌8B32          ┌mov esi,dword ptr ds:[edx]
004010D2  .  │3B31          │cmp esi,dword ptr ds:[ecx]
004010D4  .,v│75 1A         │jnz short vm_vmp.004010F0
004010D6  .  │83E8 04       │sub eax,0x4
004010D9  .  │83C1 04       │add ecx,0x4
004010DC  .  │83C2 04       │add edx,0x4
004010DF  .  │83F8 04       │cmp eax,0x4
004010E2  .^ └73 EC         └jnb short vm_vmp.004010D0
004010E4  .   8A01          mov al,byte ptr ds:[ecx]
004010E6  .   3A02          cmp al,byte ptr ds:[edx]
004010E8  .   75 06         jnz short vm_vmp.004010F0
堆栈 ds:[0012FF2C]=50425356
esi=0040D070 (vm_vmp.0040D070)
跳转来自 004010E2

地址      HEX 数据                                          ASCII
0012FF2C  56 53 42 50 4B 5F 56 6F 65 7F 61 6F 0D 7C 71 6F  VSBPK_Voe■ao.|qo
0012FF3C  63 7F 6F 63 79 0D 7C 62 49 00 40 00 02 CF 3C CE  c■ocy.|bI.@.■?■
0012FF4C  1A AD 0E CE 9E CF 3C CE 8E 2F 0E CE 8A E3 3A 47  ■?滎?蟒/■蠁?G
0012FF5C  C8 CF CF CF 45 6C 40 00 66 6C 61 67 00 00 40 00  认舷El@.flag..@.
0012FF6C  63 6F 6F 6C 00 2F 40 00 C4 91 5E 77 C0 FF 12 00  cool./@.牄^w?■.
0012FF7C  8C 16 40 00 01 00 00 00 20 2F BA 00 68 2F BA 00  ?@.■... /?h/?
0012FF8C  7C 91 5E 77 28 02 93 7C FF FF FF FF 00 E0 FD 7F  |憓w(■搢ÿÿÿÿ.帻■
0012FF9C  06 02 00 00 D9 18 41 00 00 00 00 00 8C FF 12 00  ■■..?A.....?■.
```

由此用py写出反函数：

```
enc = [0x56,0x53,0x42,0x50,0x4B,0x5F,0x56,0x6F,0x65,0x7F,0x61,0x6F,0x0D,0x7C,0x71,0x6F,0x63,0x7F,0x6F,0x63,0x79,0x0D,0x7C,0x62,0x49]
dec=[]
for i in range(len(enc)):
    dec.append(chr(((enc[i])^0x33)+3))
print ''.join(dec)

#hctf{oh_YOU_ARE_SO_SMART}
```

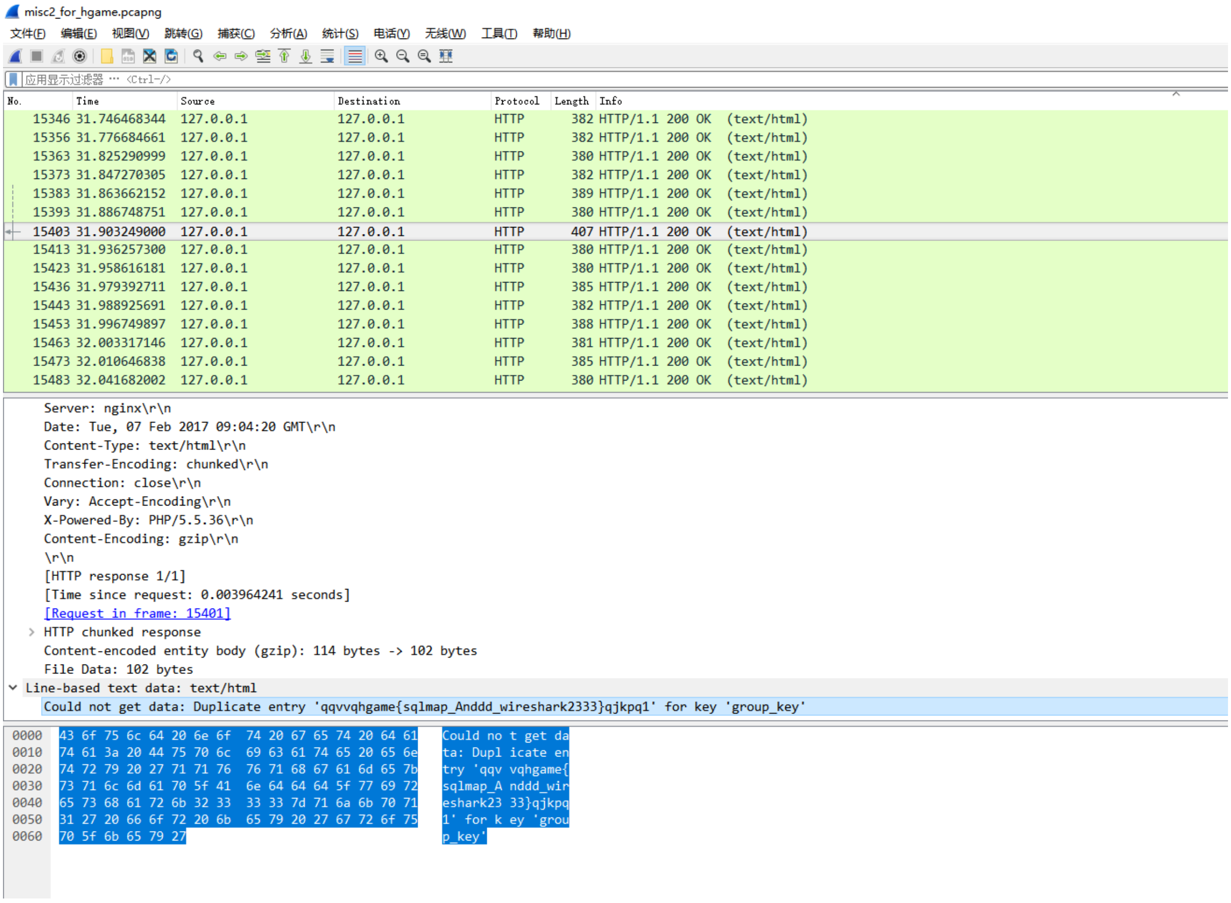
得flag：`hctf{oh_YOU_ARE_SO_SMART}`

后话：如果你知道vs编译出来的程序的用户函数一般都从data段最前面开始，而且你有成功猜到这道题的主函数在0x401000处的话，这道题会简单很多：首先下一个退出断点：BP ExitProcess，运行后断下，ctrl+G来到401000，发现代码完整，接着在数据窗口ctrl+G来到401000，对401000下硬件执行断点，重新载入，f9运行，程序成功断在401000处，然后和上面一样，很快就解得了flag。

Misc

来看看自己是怎么日自己的

放完图就跑。



```
No.        Time          Source         Destination    Protocol  Length Info
    15346 31.746468344  127.0.0.1      127.0.0.1      HTTP        382 HTTP/1.1 200 OK  (text/html)
    15356 31.776684661  127.0.0.1      127.0.0.1      HTTP        382 HTTP/1.1 200 OK  (text/html)
    15363 31.825290999  127.0.0.1      127.0.0.1      HTTP        380 HTTP/1.1 200 OK  (text/html)
    15373 31.847270305  127.0.0.1      127.0.0.1      HTTP        382 HTTP/1.1 200 OK  (text/html)
    15383 31.863662152  127.0.0.1      127.0.0.1      HTTP        389 HTTP/1.1 200 OK  (text/html)
    15393 31.886748751  127.0.0.1      127.0.0.1      HTTP        380 HTTP/1.1 200 OK  (text/html)
    15403 31.903249000  127.0.0.1      127.0.0.1      HTTP        407 HTTP/1.1 200 OK  (text/html)
    15413 31.936257300  127.0.0.1      127.0.0.1      HTTP        380 HTTP/1.1 200 OK  (text/html)
    15423 31.958616181  127.0.0.1      127.0.0.1      HTTP        380 HTTP/1.1 200 OK  (text/html)
    15436 31.979392711  127.0.0.1      127.0.0.1      HTTP        385 HTTP/1.1 200 OK  (text/html)
    15443 31.988925691  127.0.0.1      127.0.0.1      HTTP        382 HTTP/1.1 200 OK  (text/html)
    15453 31.996749897  127.0.0.1      127.0.0.1      HTTP        388 HTTP/1.1 200 OK  (text/html)
    15463 32.003317146  127.0.0.1      127.0.0.1      HTTP        381 HTTP/1.1 200 OK  (text/html)
    15473 32.010646838  127.0.0.1      127.0.0.1      HTTP        385 HTTP/1.1 200 OK  (text/html)
    15483 32.041682002  127.0.0.1      127.0.0.1      HTTP        380 HTTP/1.1 200 OK  (text/html)
```

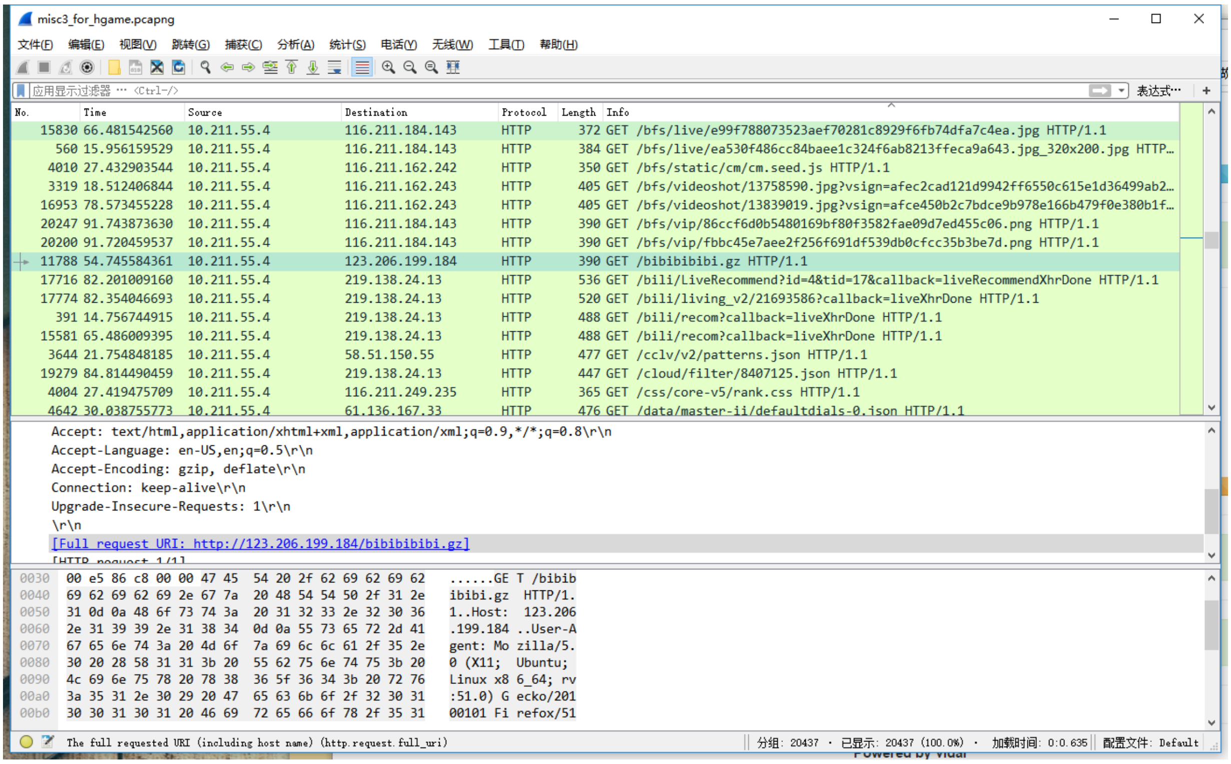
```
    Server: nginx\r\n
    Date: Tue, 07 Feb 2017 09:04:20 GMT\r\n
    Content-Type: text/html\r\n
    Transfer-Encoding: chunked\r\n
    Connection: close\r\n
    Vary: Accept-Encoding\r\n
    X-Powered-By: PHP/5.5.36\r\n
    Content-Encoding: gzip\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.003964241 seconds]
    [Request in frame: 15401]
  > HTTP chunked response
    Content-encoded entity body (gzip): 114 bytes -> 102 bytes
    File Data: 102 bytes
v Line-based text data: text/html
    Could not get data: Duplicate entry 'qqvvqhgame{sqlmap_Anddd_wireshark2333}qjkpq1' for key 'group_key'
```

```
0000  43 6f 75 6c 64 20 6e 6f  74 20 67 65 74 20 64 61   Could no t get da
0010  74 61 3a 20 44 75 70 6c  69 63 61 74 65 20 65 6e   ta: Dupl icate en
0020  74 72 79 20 27 71 71 76  76 71 68 67 61 6d 65 7b   try 'qqv vqhgame{
0030  73 71 6c 6d 61 70 5f 41  6e 64 64 64 5f 77 69 72   sqlmap_A nddd_wir
0040  65 73 68 61 72 6b 32 33  33 33 7d 71 6a 6b 70 71   eshark23 33}qjkpq
0050  31 27 20 66 6f 72 20 6b  65 79 20 27 67 72 6f 75   1' for k ey 'grou
0060  70 5f 6b 65 79 27                                  p_key'
```

flag: `hgame{sqlmap_Anddd_wireshark2333}`

考眼力喽

放完图就跑。


misc3_for_hgame.pcapng
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
表达式…

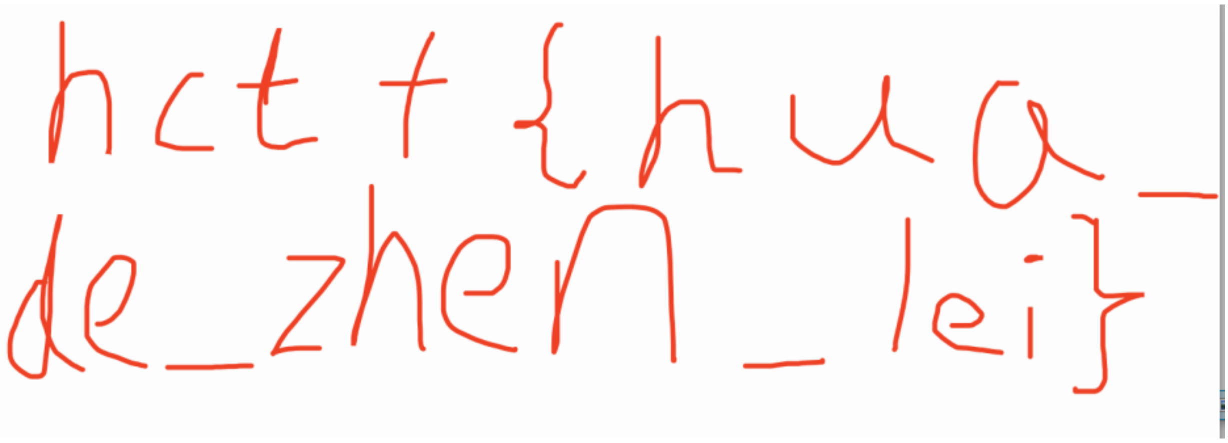
```
No.   Time         Source       Destination      Protocol  Length Info
15830 66.481542560 10.211.55.4  116.211.184.143  HTTP   372 GET /bfs/live/e99f788073523aef70281c8929f6fb74dfa7c4ea.jpg HTTP/1.1
  560 15.956159529 10.211.55.4  116.211.184.143  HTTP   384 GET /bfs/live/ea530f486cc84baee1c324f6ab8213ffeca9a643.jpg_320x200.jpg HTTP…
 4010 27.432903544 10.211.55.4  116.211.162.242  HTTP   350 GET /bfs/static/cm/cm.seed.js HTTP/1.1
 3319 18.512406844 10.211.55.4  116.211.162.243  HTTP   405 GET /bfs/videoshot/13758590.jpg?vsign=afec2cad121d9942ff6550c615e1d36499ab2…
16953 78.573455228 10.211.55.4  116.211.162.243  HTTP   405 GET /bfs/videoshot/13839019.jpg?vsign=afce450b2c7bdce9b978e166b479f0e380b1f…
20247 91.743873630 10.211.55.4  116.211.184.143  HTTP   390 GET /bfs/vip/86ccf6d0b5480169bf80f3582fae09d7ed455c06.png HTTP/1.1
20200 91.720459537 10.211.55.4  116.211.184.143  HTTP   390 GET /bfs/vip/fbbc45e7aee2f256f691df539db0cfcc35b3be7d.png HTTP/1.1
11788 54.745584361 10.211.55.4  123.206.199.184  HTTP   390 GET /bibibibibi.gz HTTP/1.1
17716 82.201009160 10.211.55.4  219.138.24.13    HTTP   536 GET /bili/LiveRecommend?id=4&tid=17&callback=liveRecommendXhrDone HTTP/1.1
17774 82.354046693 10.211.55.4  219.138.24.13    HTTP   520 GET /bili/living_v2/21693586?callback=liveXhrDone HTTP/1.1
  391 14.756744915 10.211.55.4  219.138.24.13    HTTP   488 GET /bili/recom?callback=liveXhrDone HTTP/1.1
15581 65.486009395 10.211.55.4  219.138.24.13    HTTP   488 GET /bili/recom?callback=liveXhrDone HTTP/1.1
 3644 21.754848185 10.211.55.4  58.51.150.55     HTTP   477 GET /cclv/v2/patterns.json HTTP/1.1
19279 84.814490459 10.211.55.4  219.138.24.13    HTTP   447 GET /cloud/filter/8407125.json HTTP/1.1
 4004 27.419475709 10.211.55.4  116.211.249.235  HTTP   365 GET /css/core-v5/rank.css HTTP/1.1
 4642 30.038755773 10.211.55.4  61.136.167.33    HTTP   476 GET /data/master-ii/defaultdials-0.json HTTP/1.1
```

```
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://123.206.199.184/bibibibibi.gz]
    [HTTP request 1/1]
```

```
0030  00 e5 86 c8 00 00 47 45  54 20 2f 62 69 62 69 62   ......GE T /bibib
0040  69 62 69 62 69 2e 67 7a  20 48 54 54 50 2f 31 2e   ibibi.gz  HTTP/1.
0050  31 0d 0a 48 6f 73 74 3a  20 31 32 33 2e 32 30 36   1..Host:  123.206
0060  2e 31 39 39 2e 31 38 34  0d 0a 55 73 65 72 2d 41   .199.184 ..User-A
0070  67 65 6e 74 3a 20 4d 6f  7a 69 6c 6c 61 2f 35 2e   gent: Mo zilla/5.
0080  30 20 28 58 31 31 3b 20  55 62 75 6e 74 75 3b 20   0 (X11;  Ubuntu;
0090  4c 69 6e 75 78 20 78 38  36 5f 36 34 3b 20 72 76   Linux x8 6_64; rv
00a0  3a 35 31 2e 30 29 20 47  65 63 6b 6f 2f 32 30 31   :51.0) G ecko/201
00b0  30 30 31 30 31 20 46 69  72 65 66 6f 78 2f 35 31   00101 Fi refox/51
```

The full requested URI (including host name) (http.request.full_uri) 分组: 20437 · 已显示: 20437 (100.0%) · 加载时间: 0:0.635 配置文件: Default

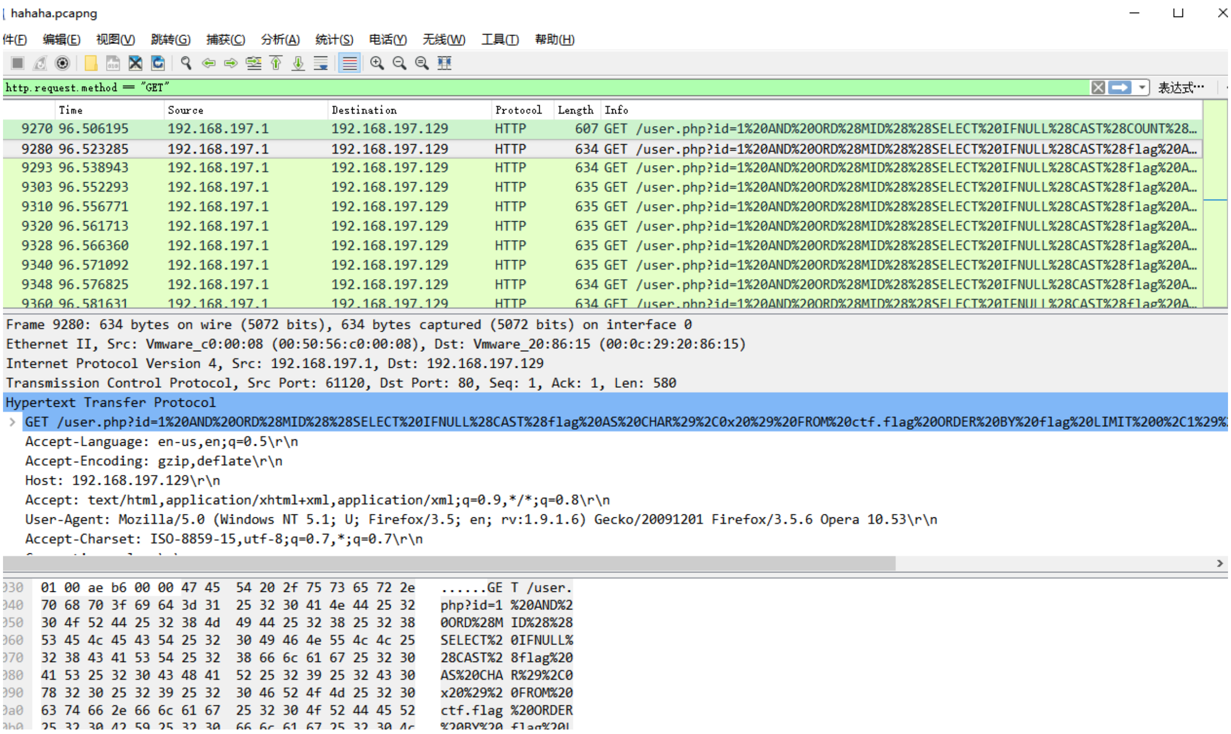
下载压缩包，解压得到一张图片：



flag：`hctf{hua_de_zhen_lei}`

正在前往翻车大道

首先过滤“GET”：


hahaha.pcapng
件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
表达式…

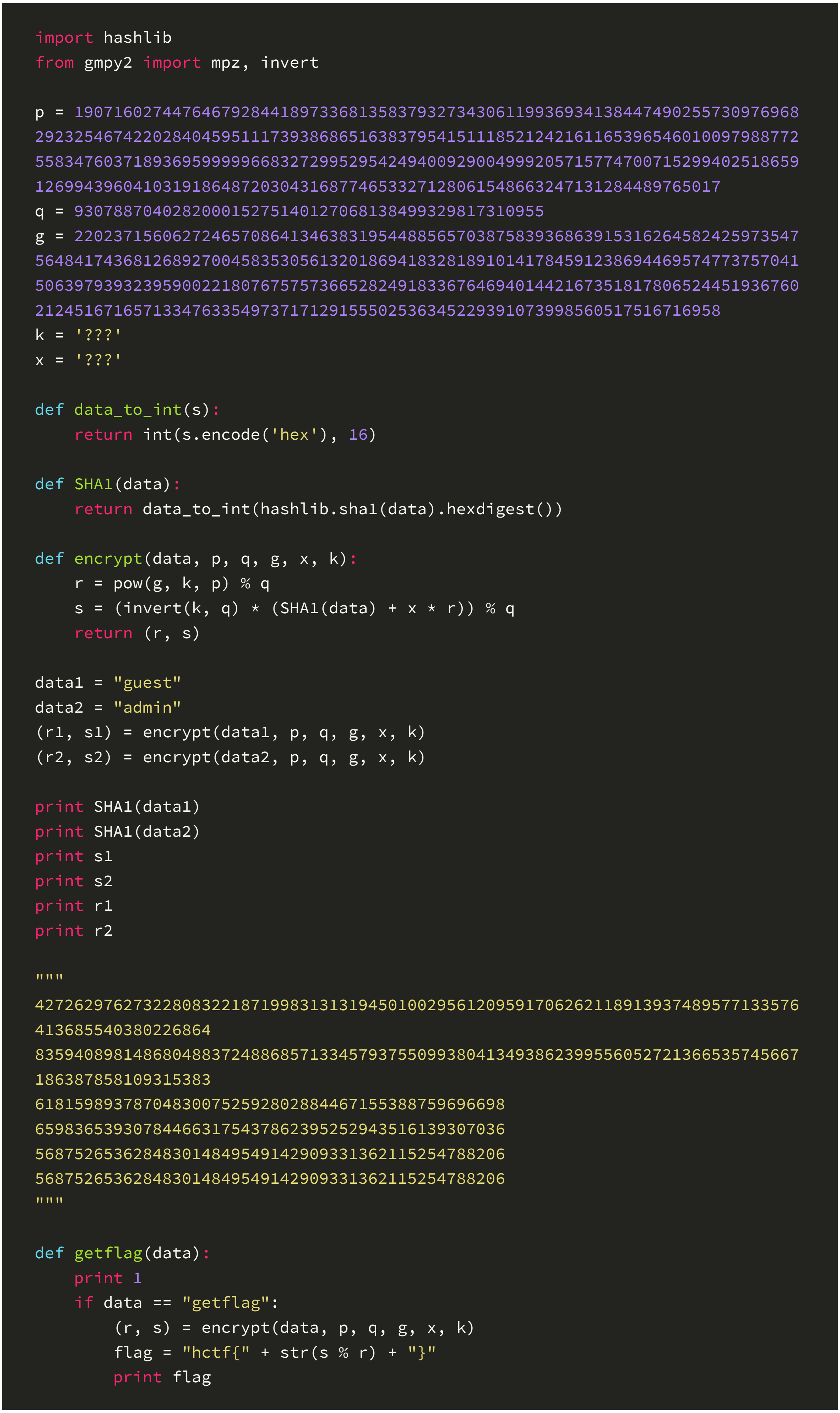
```
http.request.method == "GET"

     Time        Source          Destination      Protocol Length Info
9270 96.506195   192.168.197.1   192.168.197.129  HTTP     607 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28COUNT%28…
9280 96.523285   192.168.197.1   192.168.197.129  HTTP     634 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9293 96.538943   192.168.197.1   192.168.197.129  HTTP     634 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9303 96.552293   192.168.197.1   192.168.197.129  HTTP     635 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9310 96.556771   192.168.197.1   192.168.197.129  HTTP     635 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9320 96.561713   192.168.197.1   192.168.197.129  HTTP     635 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9328 96.566360   192.168.197.1   192.168.197.129  HTTP     635 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9340 96.571092   192.168.197.1   192.168.197.129  HTTP     635 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9348 96.576825   192.168.197.1   192.168.197.129  HTTP     634 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…
9360 96.581631   192.168.197.1   192.168.197.129  HTTP     634 GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20A…

Frame 9280: 634 bytes on wire (5072 bits), 634 bytes captured (5072 bits) on interface 0
Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_20:86:15 (00:0c:29:20:86:15)
Internet Protocol Version 4, Src: 192.168.197.1, Dst: 192.168.197.129
Transmission Control Protocol, Src Port: 61120, Dst Port: 80, Seq: 1, Ack: 1, Len: 580
Hypertext Transfer Protocol
 > GET /user.php?id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28flag%20AS%20CHAR%29%2C0x20%29%20FROM%20ctf.flag%20ORDER%20BY%20flag%20LIMIT%200%2C1%29%
   Accept-Language: en-us,en;q=0.5\r\n
   Accept-Encoding: gzip,deflate\r\n
   Host: 192.168.197.129\r\n
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
   User-Agent: Mozilla/5.0 (Windows NT 5.1; U; Firefox/3.5; en; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 Opera 10.53\r\n
   Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7\r\n

030  01 00 ae b6 00 00 47 45  54 20 2f 75 73 65 72 2e   ......GE T /user.
040  70 68 70 3f 69 64 3d 31  25 32 30 41 4e 44 25 32   php?id=1 %20AND%2
050  30 4f 52 44 25 32 38 4d  49 44 25 32 38 25 32 38   0ORD%28M ID%28%28
060  53 45 4c 45 43 54 25 32  30 49 46 4e 55 4c 4c 25   SELECT%2 0IFNULL%
070  32 38 43 41 53 54 25 32  38 66 6c 61 67 25 32 30   28CAST%2 8flag%20
080  41 53 25 32 30 43 48 41  52 25 32 39 25 32 43 30   AS%20CHA R%29%2C0
090  78 32 30 25 32 39 25 32  30 46 52 4f 4d 25 32 30   x20%29%2 0FROM%20
0a0  63 74 66 2e 66 6c 61 67  25 32 30 4f 52 44 45 52   ctf.flag %20ORDER
0b0  25 32 30 42 59 25 32 30  66 6c 61 67 25 32 30 4c   %20BY%20 flag%20L
```

发现这些包中含有flag字段，成功引起了我的注意，url解码一下，形式如下：`http://192.168.197.129/user.php?id=1 AND ORD(MID((SELECT IFNULL(CAST(flag AS CHAR),0x20) FROM ctf.flag ORDER BY flag LIMIT 0,1),1,1))>96`,一条条向下观察，发现最后一位是ascii码，大概用的是二分法。因为是flag，那八成是hctf开头，看了一下数据包，是`gcte`，第一位和第四位要加一。又因为是大于号，所以最终的字符应该是ord(i)或ord(i+1)。先一位位提出来再说，得到`gcte{ekoweq_sqk_imiecsiom}`,适当加一后拼出连贯单词，即flag：`hctf{flower_sql_injection}`

Crypto

进击的 Crypto [4]

题目不长，先粘上：



```python
import hashlib
from gmpy2 import mpz, invert

p = 19071602744764679284418973368135837932734306119936934138447490255730976968
292325467422028404595111739386865163837954151118521242161165396546010097988772
558347603718936959999966832729952954249400929004999205715774700715299402518659
1269943960410319186487203043168774653327128061548663247131284489765017
q = 930788704028200015275140127068138499329817310955
g = 22023715606272465708641346383195448856570387583936863915316264582425973547
564841743681268927004583530561320186941832818910141784591238694469574773757041
506397939323959002218076757573665282491833676469401442167351817806524451936760
2124516716571334763354973717129155502536345229391073998560517516716958
k = '???'
x = '???'

def data_to_int(s):
    return int(s.encode('hex'), 16)

def SHA1(data):
    return data_to_int(hashlib.sha1(data).hexdigest())

def encrypt(data, p, q, g, x, k):
    r = pow(g, k, p) % q
    s = (invert(k, q) * (SHA1(data) + x * r)) % q
    return (r, s)

data1 = "guest"
data2 = "admin"
(r1, s1) = encrypt(data1, p, q, g, x, k)
(r2, s2) = encrypt(data2, p, q, g, x, k)

print SHA1(data1)
print SHA1(data2)
print s1
print s2
print r1
print r2

"""
427262976273228083221871998313131945010029561209591706262118913937489577133576
413685540380226864
835940898148680488372488685713345793755099380413493862399556052721366535745667
186387858109315383
618159893787048300752592802884467155388759696698
659836539307844663175437862395252943516139307036
568752653628483014849549142909331362115254788206
568752653628483014849549142909331362115254788206
"""

def getflag(data):
    print 1
    if data == "getflag":
        (r, s) = encrypt(data, p, q, g, x, k)
        flag = "hctf{" + str(s % r) + "}"
        print flag
```

这里我做一个小小的改动，把gmpy2库改成gmpy，我的linux不知怎的就是装不上gmpy2，就拿gmpy将就一下，反正也能做题，如果有大佬知道怎么装请务必告诉我谢谢。

顺便说一下，我查资料的时候看到一种叫DSA的加密算法和他好像，虽然不知道这是不是DSA。

前排提醒：因本人还未学过离散，以下内容可能有误！！

先观察加密函数：
$r = (g^k \mod p) \mod q$
$s = (invert(k, q) * (SHA1(data) + x * r)) \mod q$
其中invert函数是求逆元。

题中给了两组加密，说明只用一组来暴力跑肯定是不科学的。如果从r入手肯定是暴力了，所以我从s入手。

两组的s相减可以发现：

$$s2 - s1 = (invert(k, q) * (sha2 - sha1)) \mod q$$

消掉了其中的x*r。我们记s = s2 - s1，sha = sha2 - sha1。
得到：$s = (y * sha) \mod q$
其中：$k * y \mod q = 1$
因此我们两边乘k，化简得到：$k = (sha \div s) \mod q$
在python中利用gmpy的`divm(sha,s,q)`解出k。

对原始的s式子两边乘k,除r得到：
$\frac{s1*k}{r} = (\frac{sha1}{r} + x) \mod q$

从而化简得到：$x = ((\frac{s1*k}{r} \mod q) - (\frac{sha1}{r} \mod q)) \mod q$

在python中利用gmpy的`(divm(s1*k,r,q) - divm(sha1,r,q)) % q`解出k。

以下为完整的python解密代码：

```python
import hashlib
from gmpy import mpz, invert,divm

p = 19071602744764679284418973368135837932734306119936934138447490255730976968
292325467422028404595111739386865163837954151118521242161165396546010097988772
558347603718936959999966832729952954249400929004999205715774700715299402518659
1269943960410319186487203043168774653327128061548663247131284489765017
q = 930788704028200015275140127068138499329817310955
g = 22023715606272465708641346383195448856570387583936863915316264582425973547
564841743681268927004583530561320186941832818910141784591238694469574773757041
506397939323959002218076757573665282491833676469401442167351817806524451936760
2124516716571334763354973717129155502536345229391073998560517516716958
sha1 = 42726297627322808322187199831313194501002956120959170626211891393748957
7133576413685540380226864
sha2 = 83594089814868048837248868571334579375509938041349386239955605272136653
5745667186387858109315383
s1 = 618159893787048300752592802884467155388759696698
s2 = 659836539307844663175437862395252943516139307036
r = 568752653628483014849549142909331362115254788206
```

```
s = s2 - s1
sha = sha2 - sha1

k = divm(sha,s,q)
x = (divm(s1*k,r,q) - divm(sha1,r,q)) % q

def data_to_int(s):
    return int(s.encode('hex'), 16)

def SHA1(data):
    return data_to_int(hashlib.sha1(data).hexdigest())

def encrypt(data, p, q, g, x, k):
    r = pow(g, k, p) % q
    s = (invert(k, q) * (SHA1(data) + x * r)) % q
    return (r, s)

def getflag(data):
    print 1
    if data == "getflag":
        (r, s) = encrypt(data, p, q, g, x, k)
        flag = "hctf{" + str(s % r) + "}"
        print flag

getflag("getflag")
```

解得flag：`hctf{88169191231439818447681393510021281730269252095}`

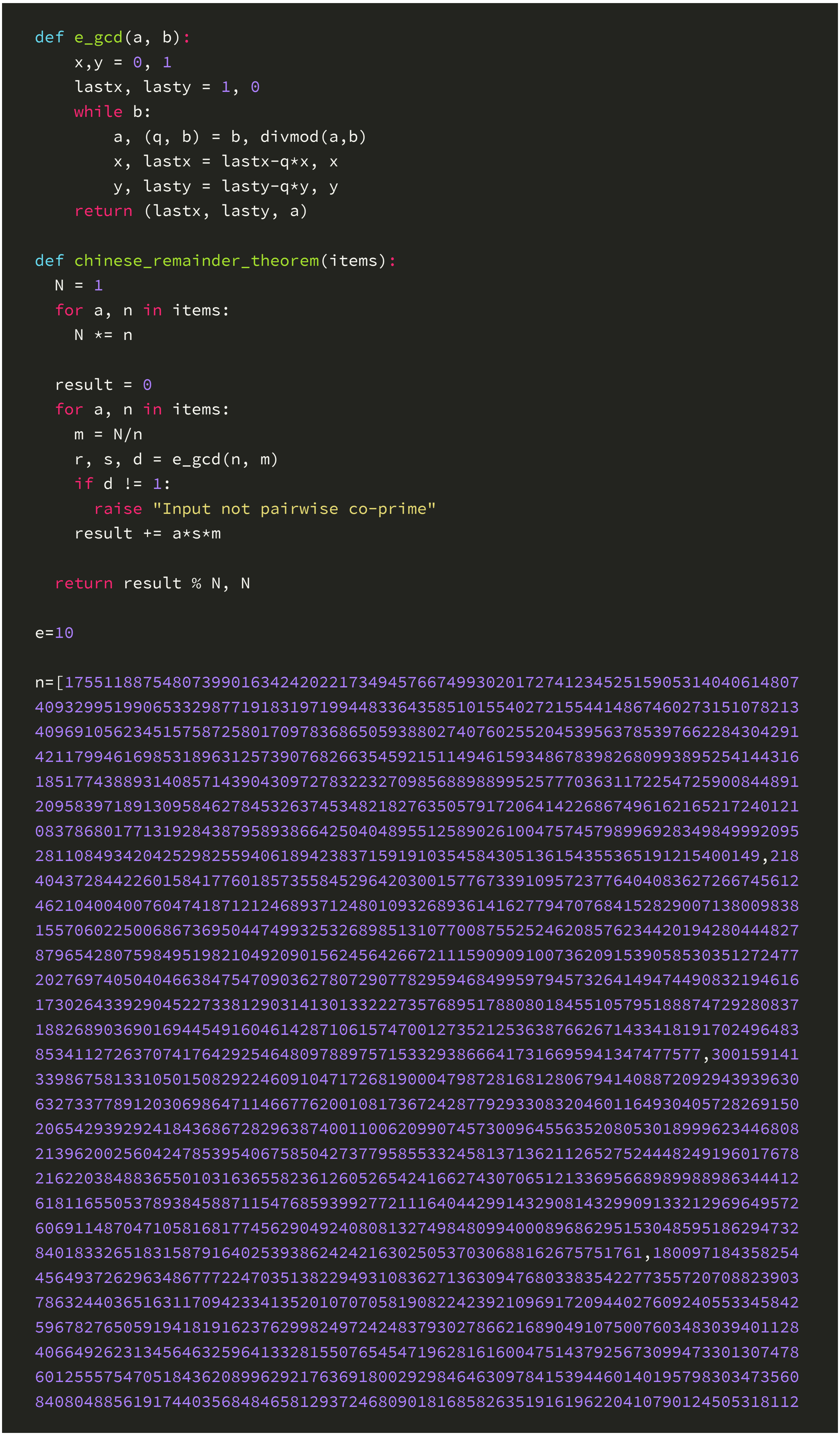
进击的 Crypto [5]

这道题给个十组RSA（别说你看不出来），e=10，如果e比较大的话应该是用同素因子分解n的方法做，但这题e很小，且给了10组，所以应该是低加密指数广播攻击了。

破解这种用中国剩余定理，不知道的百度查一下。

以下为py脚本：

```
import gmpy

def my_parse_number(number):
    string = "%x" % number
    erg = []
    while string != '':
        erg = erg + [chr(int(string[:2], 16))]
        string = string[2:]
    return ''.join(erg)
```



```
def e_gcd(a, b):
    x,y = 0, 1
    lastx, lasty = 1, 0
    while b:
        a, (q, b) = b, divmod(a,b)
        x, lastx = lastx-q*x, x
        y, lasty = lasty-q*y, y
    return (lastx, lasty, a)

def chinese_remainder_theorem(items):
  N = 1
  for a, n in items:
    N *= n

  result = 0
  for a, n in items:
    m = N/n
    r, s, d = e_gcd(n, m)
    if d != 1:
      raise "Input not pairwise co-prime"
    result += a*s*m

  return result % N, N

e=10

n=[175511887548073990163424202217349457667499302017274123452515905314040614807
409329951990653329877191831971994483364358510155402721554414867460273151078213
409691056234515758725801709783686505938802740760255204539563785397662284304291
421179946169853189631257390768266354592151149461593486783982680993895254144316
185177438893140857143904309727832232709856889889952577703631172254725900844891
209583971891309584627845326374534821827635057917206414226867496162165217240121
083786801771319284387958938664250404895512589026100475745798996928349849992095
28110849342042529825594061894238371591910354584305136154355365191215400149,218
404372844226015841776018573558452964203001577673391095723776404083627266745612
462104004007604741871212468937124801093268936141627794707684152829007138009838
155706022500686736950447499325326898513107700875525246208576234420194280444827
879654280759849519821049209015624564266721115909091007362091539058530351272477
202769740504046638475470903627807290778295946849959794573264149474490832194616
173026433929045227338129031413013322273576895178808018455105795188874729280837
188268903690169445491604614287106157470012735212536387662671433418191702496483
85341127263707417642925464809788975715332938666417316695941347477577,300159141
339867581331050150829224609104717268190004798728168128067941408872092943939630
632733778912030698647114667762001081736724287792933083204601164930405728269150
206542939292418436867282963874001100620990745730096455635208053018999623446808
213962002560424785395406758504273779585533245813713621126527524448249196017678
216220384883655010316365582361260526542416627430706512133695668989988986344412
618116550537893845887115476859399277211164044299143290814329909133212969649572
606911487047105816817745629049240808132749848099400089686295153048595186294732
84018332651831587916402539386242421630250537030688162675751761,180097184358254
456493726296348677722470351382294931083627136309476803383542277355720708823903
786324403651631170942334135201070705819082242392109691720944027609240553345842
596782765059194181916237629982497242483793027866216890491075007603483039401128
406649262313456463259641332815507654547196281616004751437925673099473301307478
601255575470518436208996292176369180029298464630978415394460140195798303473560
840804885619174403568484658129372468090181685826351916196220410790124505318112
```

```
371681056343821086341358808870935256112631869252130279133374547069197549235937
14509998062073450679720905569489304398625255143801533277,270907364223939911892
```

```
574381267435095525800379459074825852392946044698602079444547807655963514518190
9093786617716473870604463315696183050495143502762614767485881979217577416,4586
033930015814975553487614321341290358072960539564849589758593801587823414394351
862074093343255479635303199078561905260793645869942622136164615776822867275172
888812989071016687658023005838776181437470204528335444413852561895603377955959
308318399246963213964044984235974352824060625970587119586612774322385216810264
819426458641852166009899106655036755913322283219739741020656812034348404092018
340277026844114272699170323014050087289715403989671918767712613414346320222745
328800096615947138667139053339483278319521201983885423491905171167675501563556
563890821756892474999549635650486944010346554641603343238490572689,12012575342
366442210994368605032582129674485327006093552902983877957202172783938071684841
031902396156899396879136388606996807608296316795628987877357547213450360953742
947781700549506469226564975927361748596520878542651668963131021269851928174681
007347771519569914690980744941965086762285630082781013781879511681353119117280
354102271439057204066405072328422057721895983540492150402309552609035008272366
176657384174081229798739099558539083074489500359062010164358300087124447037922
844709870430218019673914305742939927888838599616520300011913022792060392260618
046807788302643742874847024153198113792628046678578753632923,42490059113248986
304587234911515768101986190783329729114201666451272585985199504485076409043793
428174864262620548891993693852578125247597612455889736515782112913443266193175
593773603195077277295350831432933357995881969762419492180378337478395452950302
591152796680658865604178772040546844950471811170756034686722922688312737359569
640676268379752185398406676418856214266079419503063301988971876837546873796118
442784715727918163979506343889072333121179206348079897340111327183343348505921
116516313231309334380070089834897826018273213961681046684933326265589118387219
98173148508994771473224050827471186478106896949413831,185571098538984059749247
695501053456737030674575378136072521514699331407607983785742444395593721004289
714000703511731403488236527558256014332688323635758961373642880699843141513461
372061155183095973892688751505496963768132777785145915326157942946370446266839
313675101816123836819144217022615922332654559500232524076806040410466296677032
071331468536189891875977491326505140385842804177911344265968483694176260395337
408275545521177275015658410650778353989518267478553935527314974389832742015997
401897436309491746107221186590196309646699014386900280702137108333356225735264
03588737092869273969581607637522572757015237506,192215313427138012199714204550
986663651248101695127110304440639423336945623641141723615521391765824761681342
199373934683913465350979657637859164840126574010950668048579009414827726871594
714839081076698924196260596801687621199540682887193035914985040506046286272996
711602252763556365199615781184139130010587441645314706102791687234020094699593
777941387152048622399735614947962136247693589067261867697116899628089378285533
994031737620345489743359601077006389102316587956391679067007701207419744184951
968278304625421300809655521024161805176003978970651137882790960667925338071041
87868678848857780735334749865813019681909,204386671853292730145412701752183594
540172717376035248822051702849886177726280703656616518928573056937508340623421
292158865409842129008826955135210183068842990360088069104660790239656677500569
414831560561643771311542422359471018667784486727321844335047925327923196006184
571189826246118597005700213852326295226083419116943876496475852687736144227766
444859612945149641569577678187926673205792058333699976663393858265624060797375
975602103248926238119076282508139261368981779276484440987711624022385237631140
416496359504034717613949446131949528534860856258990167162006362068852424952370
0500144604558899272202564535295466,9018406452041117867927204674649220929160509
942604290572303644183451237944912470236367308266319079349289625873447647200474
598755869642999213142758825809522474810817609751187533346902667225032452694187
162262683056445775455963994237202296104812386374636270188433521927503415507963
477240913270879794141170568393025367891003921090015993944946205723601349149601
256727433716155121883420271498265967234259068535172501445643752691860997480636
101699340740265505767091920668218018002976632111256430655619061247292025654740
889077014179258647084698229733160071306390020603636210736363198302407289195286
```

```
953980520285032528614492926,17978671919734595201524186618868659656036765546673
```

解得flag：`hctf{Hastad's_broadcast_attack_is_interesting}`