强网杯2021 qwb misc WP 5题

原创

是Mumuzi 于 2021-06-16 22:21:22 发布 1179 收藏 5

分类专栏： ctf

本文链接：https://blog.csdn.net/qq_42880719/article/details/117968361

BlueTeaming、ISO1995、CipherMan、ExtremelySlow、EzTime

本来是说昨天发的，忘了……

然后取证题，比较偏向爆破出的flag；如果是想知道原理等可以去看看空白师傅的

WP：https://mp.weixin.qq.com/s/ltalgLxcXCjvAhyyP3On9Q

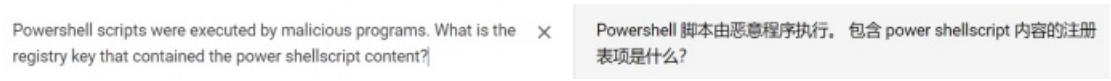
BlueTeaming

下载解压，得到一个无后缀，winhex查看是7z文件头，添加.7z，解压

得到一个5个G的memory.dmp文件

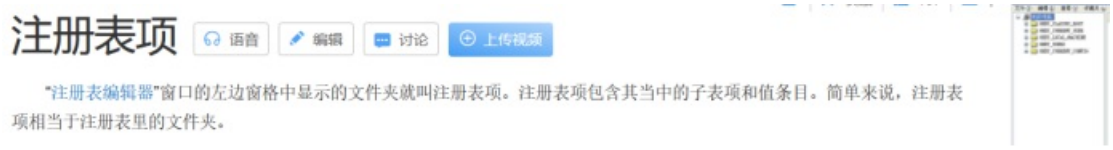
是一个内存镜像，常见的内存镜像文件有raw、vmem、dmp、img，这里也正好满足

题目问的是



所以要查找有关powershell的hive

根据百度可知，注册表项是左边的路径(HKEY)



现在开始对镜像下手，首先查看镜像的基本信息
volatility -f memory.dmp imageinfo

```
mumuzi@kali:~/桌面$ volatility -f memory.dmp imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x6
4_24000, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_24000, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/mumuzi/桌面/memory.dmp)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf80002be3120L
          Number of Processors : 4
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff80002be5000L
                KPCR for CPU 1 : 0xfffff88002f00000L
                KPCR for CPU 2 : 0xfffff88002f7e000L
                KPCR for CPU 3 : 0xfffff880009b1000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2020-11-26 13:00:41 UTC+0000
     Image local date and time : 2020-11-26 22:00:41 +0900
```

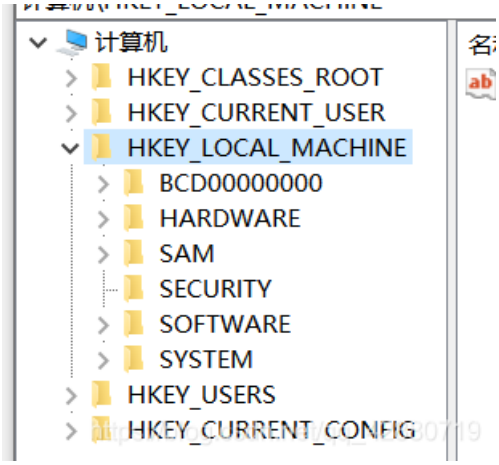

然后直接看hive

```
mumuzi@kali:~/桌面$ volatility -f memory.dmp --profile=Win7SP1x64 hivelist
Volatility Foundation Volatility Framework 2.6
Virtual            Physical           Name
------------------ ------------------ ----
0xfffff8a00000f010 0x00000000a9107010 [no name]
0xfffff8a000024010 0x00000000a9612010 \REGISTRY\MACHINE\SYSTEM
0xfffff8a000057410 0x00000000a9745410 \REGISTRY\MACHINE\HARDWARE
0xfffff8a0001052e0 0x00000000595002e0 \SystemRoot\System32\Config\DEFAULT
0xfffff8a0002d7010 0x00000000a103c010 \SystemRoot\System32\Config\SOFTWARE
0xfffff8a000347010 0x00000000a57be010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8a00200e010 0x0000000081c17010 \SystemRoot\System32\Config\SECURITY
0xfffff8a00203a010 0x0000000081a77010 \SystemRoot\System32\Config\SAM
0xfffff8a002149010 0x0000000081565010 \??\C:\Windows\ServiceProfiles\NetworkService\NTUS
ER.DAT
0xfffff8a0021d5010 0x0000000081255010 \??\C:\Windows\ServiceProfiles\LocalService\NTUSER
.DAT
0xfffff8a002ae4410 0x000000007d278410 \??\C:\Windows\System32\config\COMPONENTS
0xfffff8a00891f010 0x000000006f172010 \??\C:\System Volume Information\Syscache.hve
0xfffff8a0089ff010 0x0000000075c42010 \??\C:\Users\nyong\ntuser.dat
0xfffff8a008bf1010 0x000000002b8a7010 \??\C:\Windows\AppCompat\Programs\Amcache.hve
0xfffff8a008e1b010 0x0000000061e3c010 \??\C:\Users\nyong\AppData\Local\Microsoft\Windows
\UsrClass.dat
```

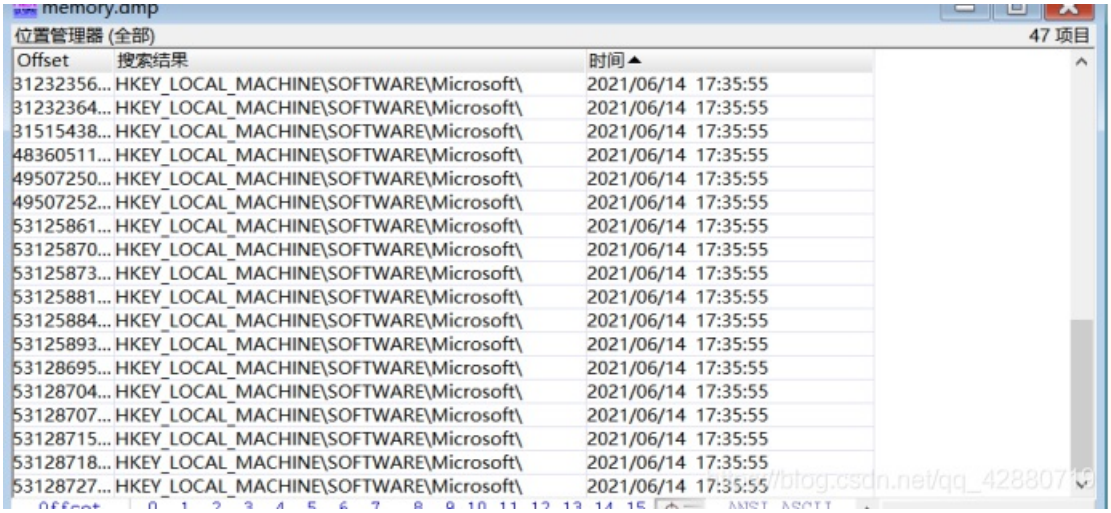

因为是powershell恶意程序，所以这里优先考虑software。
尝试用dumpregistry，笑死，根本找不到插件

```
mumuzi@kali:~/桌面$ volatility -f memory.dmp --profile=Win7SP1x64 --dump-dir=./
Volatility Foundation Volatility Framework 2.6
Usage: Volatility - A memory forensics analysis platform.
```

你说百度搜吧，没搜到。谷歌吧，没找到。看万能的gayhub吧，也没有。
没办法，直接winhex找了呗，反正是HKEY，再加上怀疑SOFTWARE，就直接找吧

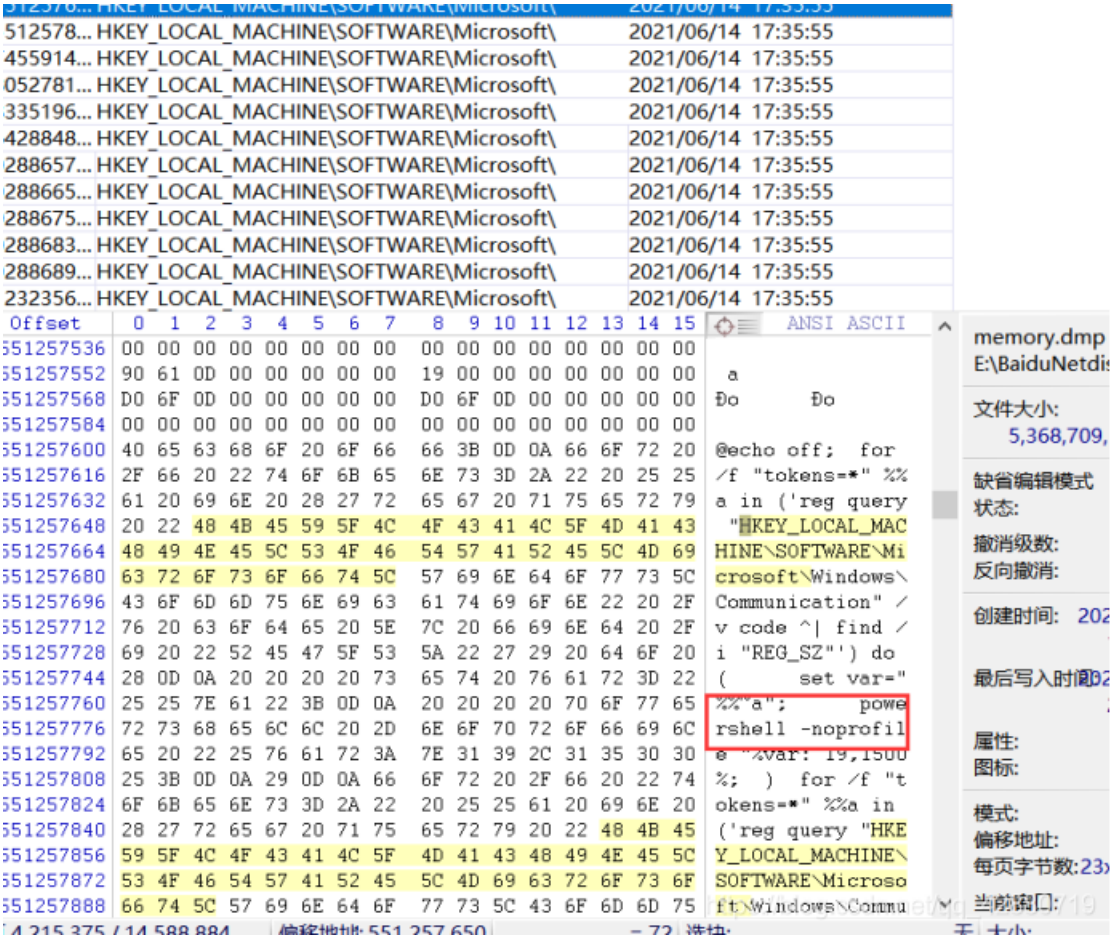


对比了一下自己的计算机，winhex直接搜HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft



一共47个项目

怎么找也是非常简单，找到周围有“powershell”的字样，尝试提交就行了。



这不

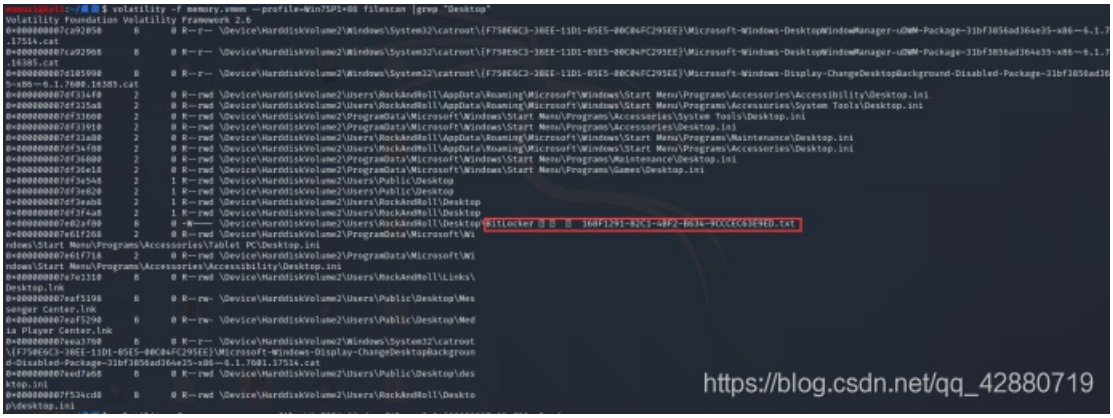
然后提交HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Communication

对了

CipherMan

解压得到一个无后缀，加上.7z解压，memory添加上.vmem后缀，然后volatility

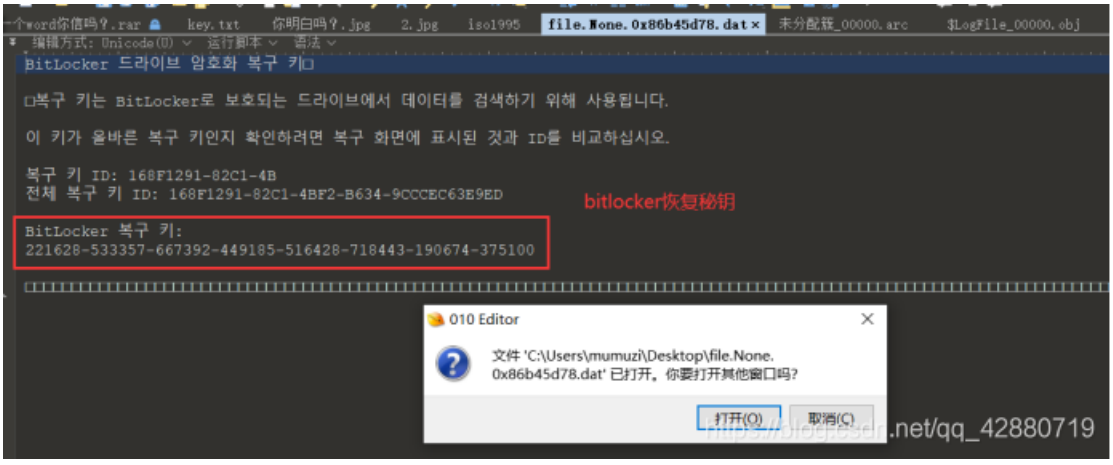
```
mumuzi@kali:~/桌面$ volatility -f memory.vmem imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1
x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/mumuzi/桌面/memory.vmem)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82d72c28L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82d73c00L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2018-08-06 08:41:18 UTC+0000
     Image local date and time : 2018-08-06 17:41:18 +0900
```

Dump出来

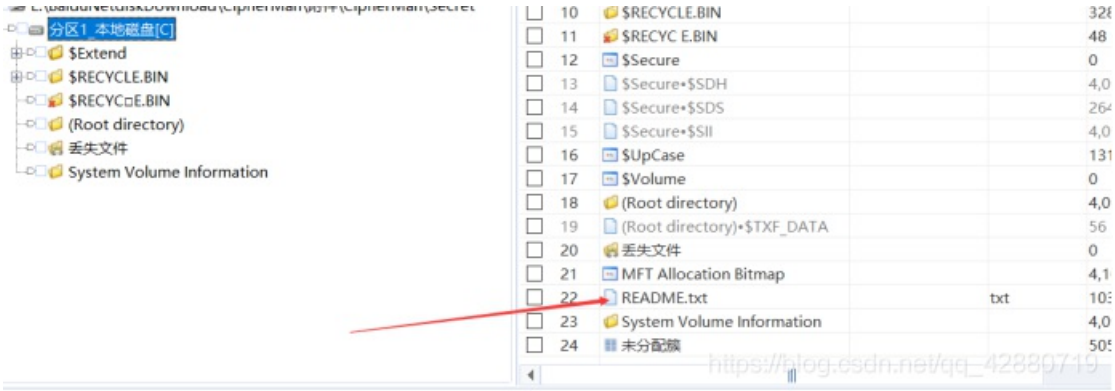
```
mumuzi@kali:~/桌面$ volatility -f memory.vmem --profile=Win7SP1x86 dumpfiles -Q 0x000000007e02af80 -D ./
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7e02af80   None   \Device\HarddiskVolume2\Users\RockAndRoll\Desktop\
BitLocker 복구 키  168F1291-82C1-4BF2-B634-9CCCEC63E9ED.txt
```

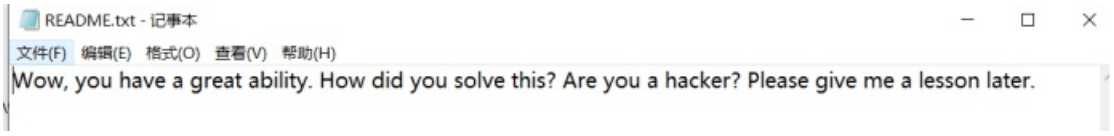
拖到010里面看



然后bitlocker解密，我用取证大师


卸载设备(U)
挂载为本地磁盘
备注
制作镜像文件(M)
制作L01逻辑证据文件(L)
重置磁盘数据源
重置镜像数据源
哈希值计算(C)
注册表解析(A)
导出当前文件夹(F)
导出所有勾选文件(Q)
加入摘录(B)
扫描磁盘结构
RAID重组
时区设置
EFS
FileVault2
BitLocker解密
Truecrypt解密
数据源解密
Luks解密
卷影分析
FAT32高级恢复
上传勾选文件
BitLocker解密
解密方式
密码
恢复密钥串
启动密钥文件
内存密钥文件
清除密钥
恢复密钥串
恢复密钥标记
168F1291-82C1-4BF2-B634-9CCCEC63E9ED
恢复密钥串
221628-533357-667392-449185-516428-718443-190674-3
恢复密钥文件
浏览
确定
取消


分区1_本地磁盘[C]
$Extend
$RECYCLE.BIN
$RECYC□E.BIN
(Root directory)
丢失文件
System Volume Information
10 $RECYCLE.BIN
11 $RECYC E.BIN
12 $Secure
13 $Secure:$SDH
14 $Secure:$SDS
15 $Secure:$SII
16 $UpCase
17 $Volume
18 (Root directory)
19 (Root directory):$TXF_DATA
20 丢失文件
21 MFT Allocation Bitmap
22 README.txt txt
23 System Volume Information
24 未分配簇


README.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Wow, you have a great ability. How did you solve this? Are you a hacker? Please give me a lesson later.

ISO1995

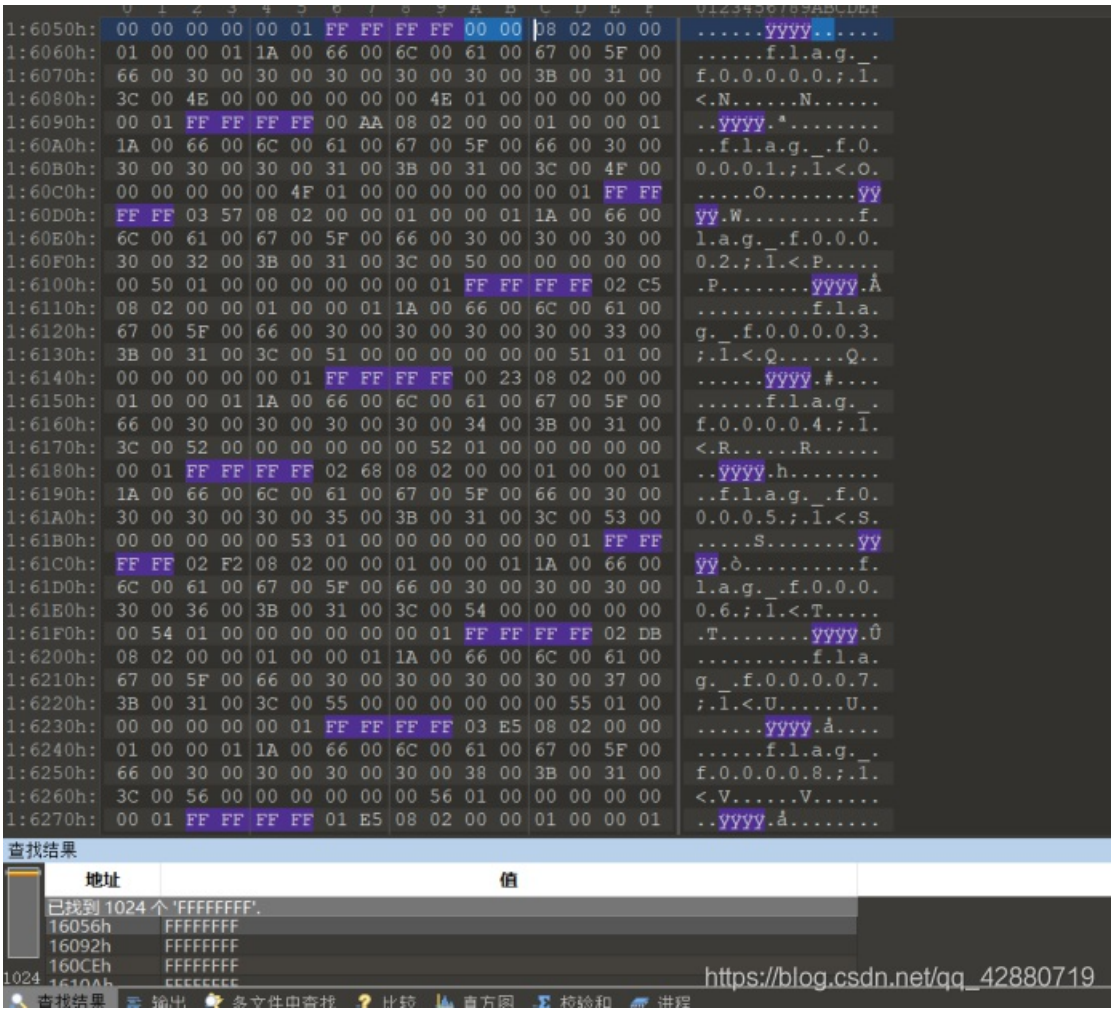
这题很绕思路，这比赛跟试错比赛似的。

首先是用UltraIISO打开，把1024个文件给下载出来。

然后尝试按照时间排序，这里就不贴脚本了(反正是错误思路)得到一串文本，有95个字符，UUencode也不对，想尝试base95发现不会写脚本……

然后重新查看iso1995，找到flag的位置，发现前面有一串FFFFFFFF

并且经过查找，一共1024个，而且后面的跟在FFFFFFFF后面的数字也在变化，并且随便看了十几个，发现总值不超过1024，猜测用这个当做那1024个提取出来的文件的顺序。



首先写脚本将FF FF FF FF后面的俩个字节提取出来

```
f = open("iso1995.iso",'rb').read()
f1 = open("flag",'wb')

def find_all_indexes(input_str, search_str):
    l1 = []
    length = len(input_str)
    index = 0
    while index < length:
        i = input_str.find(search_str, index)
        if i == -1:
            return l1
        l1.append(i)
        index = i + 1
    return l1

s = find_all_indexes(f,b'\xff\xff\xff\xff')
for i in s:
    f1.write(f[i+4:i+6])
print("done")
```

然后把10进制的值提取出来（注释掉上面那段）

```
s = [""]*1024
f = "000000AA035702C50023026802F202DB03E501E5010102A80236026703F2009E002A00E0039801B0028A006D03D1019D01CB031B031
902A0025900A1022D030500510231003C02C702E301990235037E0185012E007400A2006C021602FC01B401AA026902CE00FD00B601D3004
903BC02F000F2016503A4032501EE03B403DE02DE03F80147024F033C0263001C03F4009702EC0091000C00BF0168023301A4038C024E024
D03C50086037C001202C101110126036801C7000702F3007000BB0304036B02D3004103AF0130017C00340397006100810 2A301A80092012
A002C01350323028E00F902D7010901A702F8032701EF000F03C8020E014A02600344034300C4022C0099026D008A030103F10273031F009
D00F501FA010D037A036101E10307021502E201B702250310003502430280 2D8010803EF01F5030A033901700107017701000 05C021C016
900630102 03E1039C00DA037D01BC014002A5034900D8033A00C1004403CF029101FF03D802390338030B010602B303FB03F0007701CD025
600EC004C03EB000101620266035B0298000A02CC0248011500A8034D025401D703E9035A02270065016B03AC03830039019503EA030D03A
E02820078001501B8011D018600F600180148038E00F4013701F0012203CD011E027201CE01F103B8022B02AE00BE03850133033E020C021
003A901BD00ED03340330021400 8F014B00D70004026501430 1A901800178034C0062030201BE02F5036600AD02AD016403330142036E03E
00345018A00E300A300FB028F03BF022F02DC039E03D90314006803B600A602D202220103004D03F9008B01F6018E03E3030000C002AA01A
10139019F0360025B01BA027600F0038D001F0006028C013203D6020B0198018802A1023801E6007C035D0064009F024A00D001BB005E02B
D019A019300F803A30362019E00D4026E03EE0352015F023201FB03C002BA03BE02EB01C1039001290399 00CC02AC02BB02A9003800E400E
1009C020A03200396020701F90192034E02B501F8032803CB024200B7038100B803E7012F019C031C03F50230023C0113006F00AE01DE007
1029601F702A2004A01280110005F024701C4034B03E603ED0318008C015A03BA03F600AB0079028100AF00EA01A20127011C0056018F02B
F0002018303A100CF013D029A031E027101380219036402610076 01AB03E800A501EB03DD01A300600096017B007E01B3035403E2012B017
103FC000901C901FC021E00B9032E02E4026C008502D902DF027A01AC03CE00E2006703650384 01DC01F40212012000CD0221021B01DA020
9013E031D005B0378016C007300CA0255008D00C20082009A0224001B0013037503470278 01D603D000EF037103EC002F03A002B103FA002
501D8004F01B102C3029F030902A4028B009302BC00D5028801B600FF011403F3030E021F002100D6018D036F032103530029015B034603D
7034F0043017303DB0160029203920337 0030017F02F70124003E02F601AF010E039D0246028701B9021A010C0394025000F303A20000011
603AD0377011A02EF02CA0153031201E8034A0311022902AF03A50359036D01A6023B021D00FA02D0033500BD025F0326037F03A801EA011
B029E03CC0201038A016A03800037013F0080029B01CA025302E90088037001C802F103B30205003A031701AD0125018701970237014E025
E036A011F020F00A9014C008E030C032B00520141035502970295027 5013A0350015702AB03FF00F7009800DD015E038701E400870055015
9013C0105019B01D1019402CF004502B000C60022026F01E001DD033B03FD01D20154005D03890024039F01DF028D03BB024500110003002
700C702BE004700C902B601B5009B006E03C3016E00B5025D0059001600080 02D0066007A037B01BF030802ED035C02CD00E500C5020802F
B028001CF025A03DA018B017A02B801040190003D03BD0089020403910289014F027401C303220151019602C203D203F7029401D900BA015
803FE03B103B501ED014D0136023A0042037403150395 00E9034200DE00D101B203400020004E015C013B00AC036302D40223002600DB017
E0123006B02DD02FF029900F1008402170014009001760206 02B4023E0372032F023D01C002E60220007202FE03C901D00053032901C6033
D027C03B903690028005002DA00D901E9001A02CB017202E80356019102EE009501C20094007F0251018401441024B03B0012102030264022
E034802FD03730249003602C8027B0226016D03C7001700400386026202700252 02E50234010F03C4010B0150029302B9016F038203AB018
902B7039B014601F2030F00CB005400C801DB035E03880083001D007B00FE0181010A003301E203C103E402FA01630358006900E702E0017
D00EE02900166027E003B02F400D3034102790179022801520 2B2036C0119033603AA0174029D011703A7023F02D6021801FD0134001902C
0000D0161032D00100149016700 2E004B020003C6004600B3005A01D5020203130131012C033200B2021301A502EA026A02D5000E007D03B
202C900EB003103D500B401E700BC039A01550367032A01560303028503510258 02D100A7027F011201EC031602C6002B00A00284004800C
E01E3024100B0025C02A601CC030602C400B1024002E703DF029C033100DF00750005006A03B7003F00D201F3026B032C00FC005801D4039
3032401AE038F018C01FE03C20182015D00E6020D0175012D003203CA02E1038B03D4027702F9025702A7031A00A403DC01C5024C021100C
30244001E022A0283027D035F00DC03A60379033F037600E80118014503D30057000B"
for i in range(1024):
    print(int(f[i*4:i*4+4],16))
    s[i] = int(f[i*4:i*4+4],16)
print(s)
print(len(s))
```

再然后，对之前那1024个文件进行重命名

```
import os
path = 'C:\\Users\\mumuzi\\Desktop\\1995'
num= 0
for file in os.listdir(path):
    os.rename(os.path.join(path,file),os.path.join(path,str(num)))
    num+=1
```

最后，将他们拼起来即可（注释掉重命名过程和第一步）

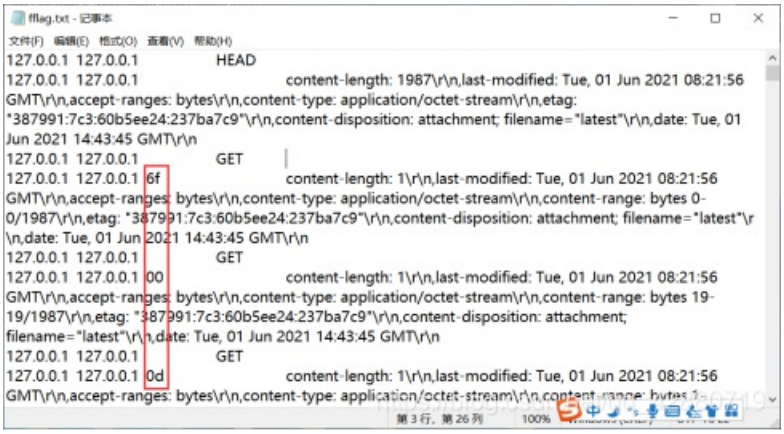
```
f = open("flag1",'wb')
for j in s:
    f1 = open("C:\\Users\\mumuzi\\Desktop\\1995\\"+str(j),"rb").read()
    f.write(f1)
print("done")
```

> FLAG{Dir3ct0ry_jYa_n41}

ExtremelySlow

HTTP里面有很多可以导出的，看了tcp流量也发现有单字节，根据标题是慢传输，再加上有个range，可以猜测是要将其提取出来然后排序

.\tshark.exe -r .\ExtremelySlow.pcapng -T fields -e ip.src -e ip.dst -e data.data -e http.request.method -e http.response.line -Y http >fflag.txt



然后按照range:bytes xx-xx的xx，按顺序，拼接起来。

因为导出的是gbk编码，python运行脚本报错，所以要把它ctrl-a ctrl-c粘贴到一个新的文本(utf-8)里，跑那个新的utf-8文本

```
f = open('fflag (2).txt','r').read()
file1 = f.split('\n')
s = []
for i in range(1987):
    s.append(0)
for file1 in file1:
    content_range_posision = file1.find('content-range')
    if (content_range_posision >= 0):
        tmp = file1[20:22]
        tmp = int(tmp, 16)
        num = file1[content_range_posision+21:content_range_posision+31]
        num = num.split('-')[0]
        num = int(num)
        s[num] = tmp
s = bytes(s)
f2 = open('latest','wb')
f2.write(s)
```

弄完观察是一个pyc文件，然后发现这个pyc不对劲，和3.7以下的文件头是对不上号的，根据pyc的文件头，在3.9以上是61开头，55是38,41是3.7以下。所以去临时装了一个python3.10a

```
 0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15      ANSI ASCII
61 0D 0D 0A 00 00 00 00  20 A0 C5 60 14 00 00 00  a         Å`
E3 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ã
00 02 00 00 00 40 00 00  00 73 0C 00 00 00 65 00       @   s    e
64 00 83 01 01 00 64 01  53 00 29 02 7A 0B 68 65  d ▌   d S ) z he
6C 6C 6F 20 77 6F 72 6C  64 4E 29 01 DA 05 70 72  llo worldN) Ú pr
69 6E 74 A9 00 72 02 00  00 00 72 02 00 00 00 FA  int© r    r    ú
09 2E 5C 74 65 73 74 2E  70 79 DA 08 3C 6D 6F 64   .\test.pyÚ <mod
75 6C 65 3E 01 00 00 00  F3 00 00 00 00           ule>    ó
```

虽然还是61开头，但是后面不是pyi0，给的脚本应该没问题。
这里找到一篇文章，是研究pyc的。

https://stackoverflow.com/questions/32562163/how-can-i-understand-a-pyc-file-content

`.pyc` files contain some metadata and a `marshal`ed `code` object; to load the `code` object and disassemble that use:

```
import dis, marshal, sys

header_sizes = [
    # (size, first version this applies to)
    # pyc files were introduced in 0.9.2 way, way back in June 1991.
    (8,  (0, 9, 2)),  # 2 bytes magic number, \r\n, 4 bytes UNIX timestamp
    (12, (3, 6)),     # added 4 bytes file size
    # bytes 4-8 are flags, meaning of 9-16 depends on what flags are set
    # bit 0 not set: 9-12 timestamp, 13-16 file size
    # bit 0 set: 9-16 file hash (SipHash-2-4, k0 = 4 bytes of the file, k1 = 0)
    (16, (3, 7)),     # inserted 4 bytes bit flag field at 4-8
    # future version may add more bytes still, at which point we can extend
    # this table. It is correct for Python versions up to 3.9
]
header_size = next(s for s, v in reversed(header_sizes) if sys.version_info >= v)

with open(pycfile, "rb") as f:
    metadata = f.read(header_size)  # first header_size bytes are metadata
    code = marshal.load(f)          # rest is a marshalled code object

dis.dis(code)
```


运行之后，会生成一大段字节码，这里我放ubuntu pastebin了
https://pastebin.ubuntu.com/p/wJ6wv8b6r2/

这里要字节码一个个反推回去
得到反推出来的python文件，这里还是放WP里吧：
里面的p = “ ”因为是做出来的了，所以已经填上了值

```
# -*- codeing = utf-8 -*-
# @Author : Mumuzi
# @File : flaggg.py
# @Software : PyCharm

import sys
from hashlib import sha256

def KSA(key):
    keylength = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i%keylength]) % 256
        S[i], S[j] = S[j], S[i]#ROT_TWO:交换两个数
    return S

def PRGA(S):
    i = 0
    j = 0

    while (1):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]

        K = S[(S[i] + S[j]) % 256]
```

```
        yield K #YIELD_VALUE

def RC4(key):
    S = KSA(key)
    return PRGA(S)

def xor(p, stream):
    return bytes(map(lambda x:x ^ stream.__next__(), p))

def bit_count(i):
    return bin(i).count("1")

if __name__ == '__main__':
    w = b'\xf6\xef\x10H\xa9\x0f\x9f\xb5\x80\xc1xd\xae\xd3\x03\xb2\x84\xc2\xb4\x0e\xc8\xf3<\x151\x19\n\x8f'
    e = b'$\r9\xa3\x18\xddW\xc9\x97\xf3\xa7\xa8R~'
    b = b'geo'
    s = b'}\xce`\xbej\xa2\x120\xb5\x8a\x94\x14{\xa3\x86\xc8\xc7\x01\x98\xa3_\x91\xd8\x82T*V\xab\xe0\xa1\x141'
    t = b"Q_\xe2\xf8\x8c\x11M}'<@\xceT\xf6?_m\xa4\xf8\xb4\xea\xca\xc7:\xb9\xe6\x06\x8b\xeb\xfabH\x85xJ3$\xdd\xde
\xb6\xdc\xa0\xb8b\x961\xb7\x13=\x17\x13\xb1"
    m = {2:115, 8:97, 11:117, 10:114}
    n = {3:119, 7:116, 9:124, 12:127}
    m |= {x:x ^ n[x] for x in n}
    m |= ((bit_count(i), i) for i in b)

    stream = RC4(list(map(lambda x:x[1], sorted(m.items()))))
    print(sha256(w).digest())
    print(xor(w, stream))

    p = b'\xe5\n2\xd6"\xf0}I\xb0\xcd\xa2\x11\xf0\xb4U\x166\xc5o\xdb\xc9\xead\x04\x15b'
    e = xor(e, stream)
    c = xor(p, stream)
    print(sha256(c).digest())

    if (sha256(c).digest() == s):
        print(xor(t, stream))
        print(c)
    else:
        print(e)
```

然后运行会提示stegosaurus
这是pyc的隐写工具，弄出来运行报错

```
File "C:\Users\mumuzi\Des
  code = marshal.load(f)
```

去查看这里

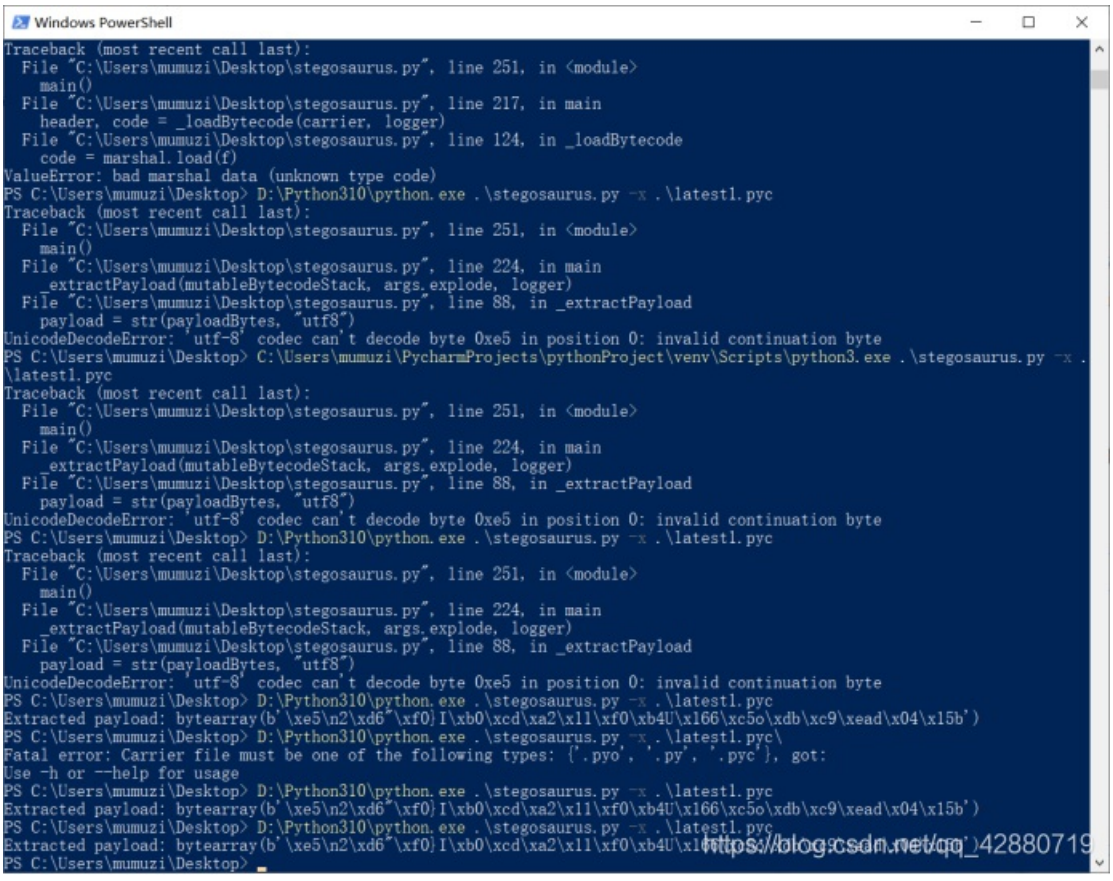
```
def _loadBytecode(carrier, logger):
    try:
        f = open(carrier, "rb")
        header = f.read(16)#12+4
        code = marshal.load(f)
        logger.debug("Read header and bytecode from carrier")
    finally:
        f.close()
```

原来是12，但是python3.9的长度比python3.7以下的header长度多4(00 00 00 00)，所以这里改成16

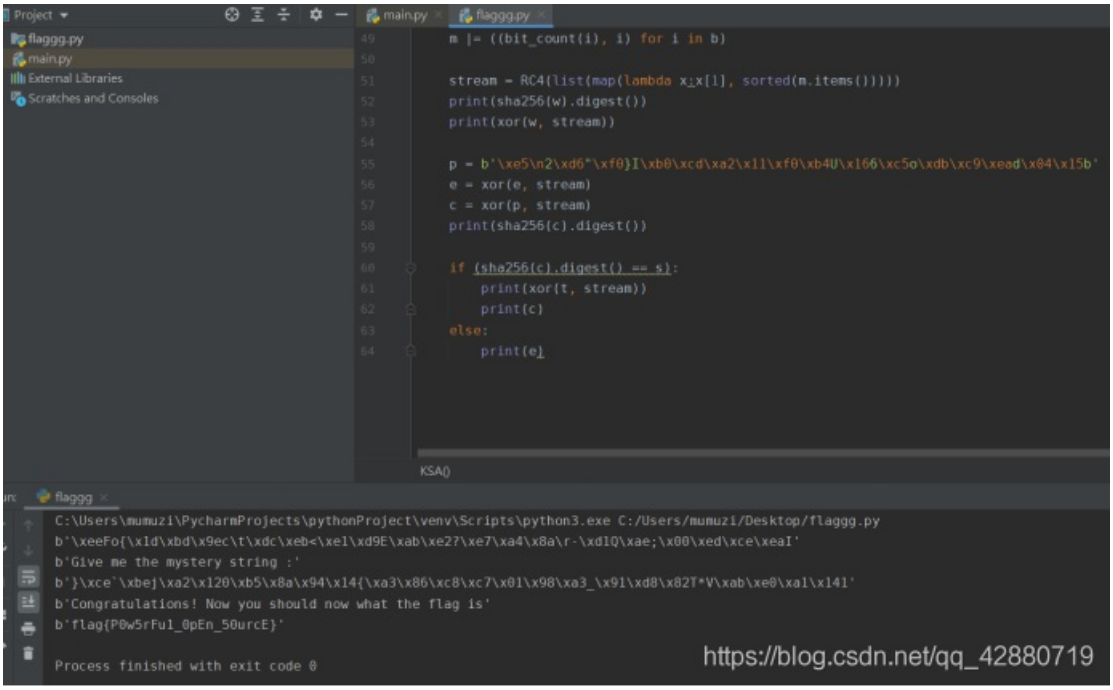
还是报错

```
88        payload = str(payloadBytes)#去掉"utf-8解码
89
```

现在运行，完美



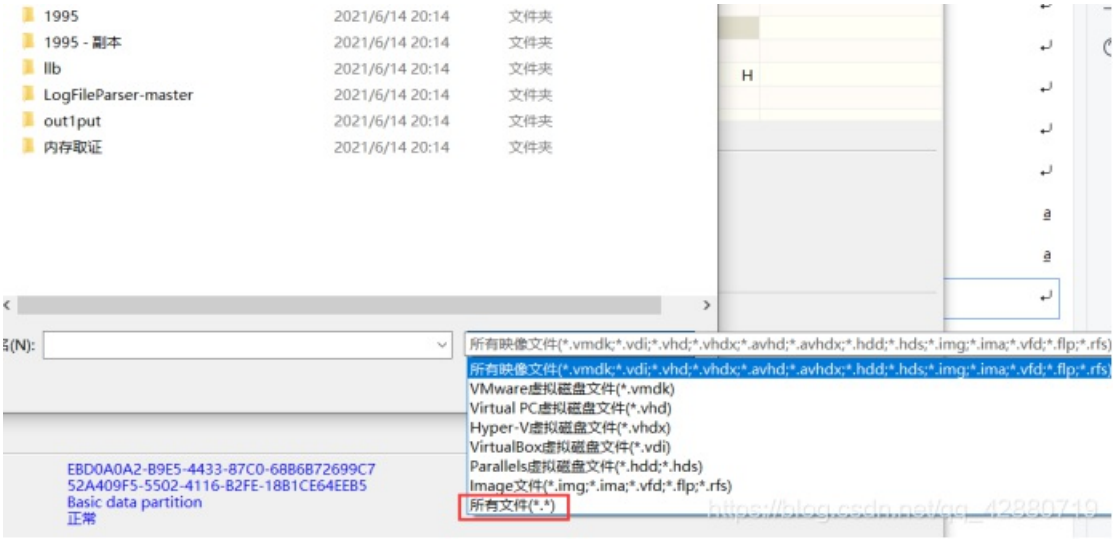
把这段密文填到p=""里面
提示congratulation!
发现只输出了一个，再把c打印出来，得到flag

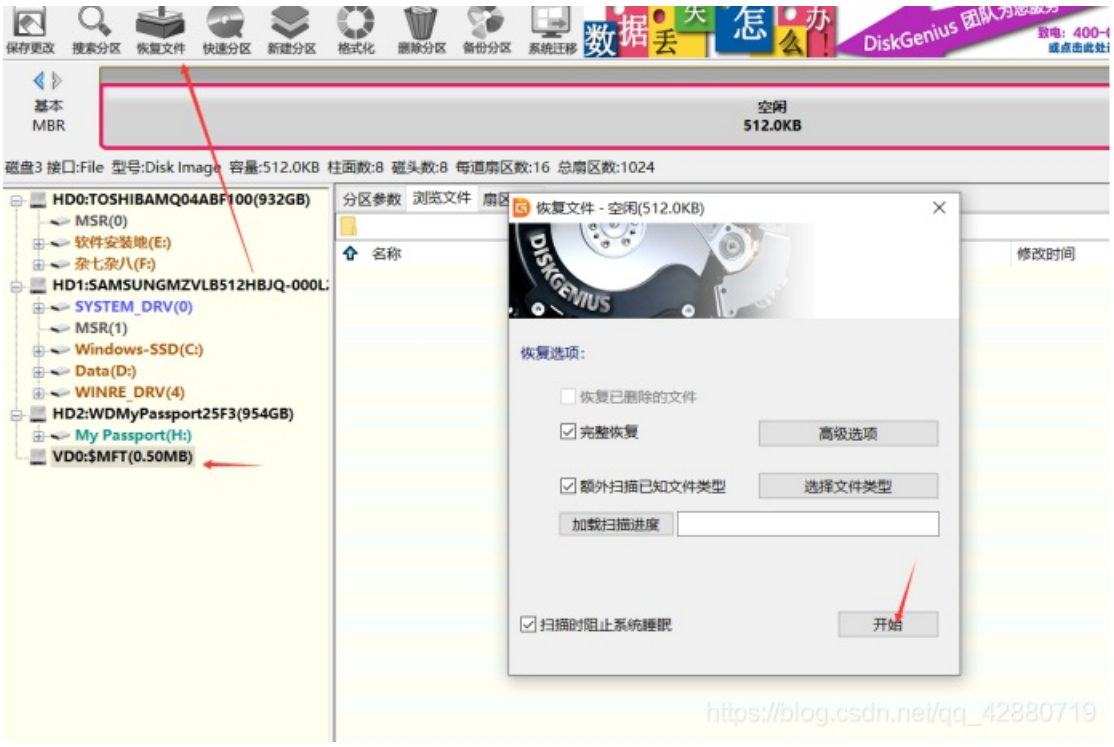


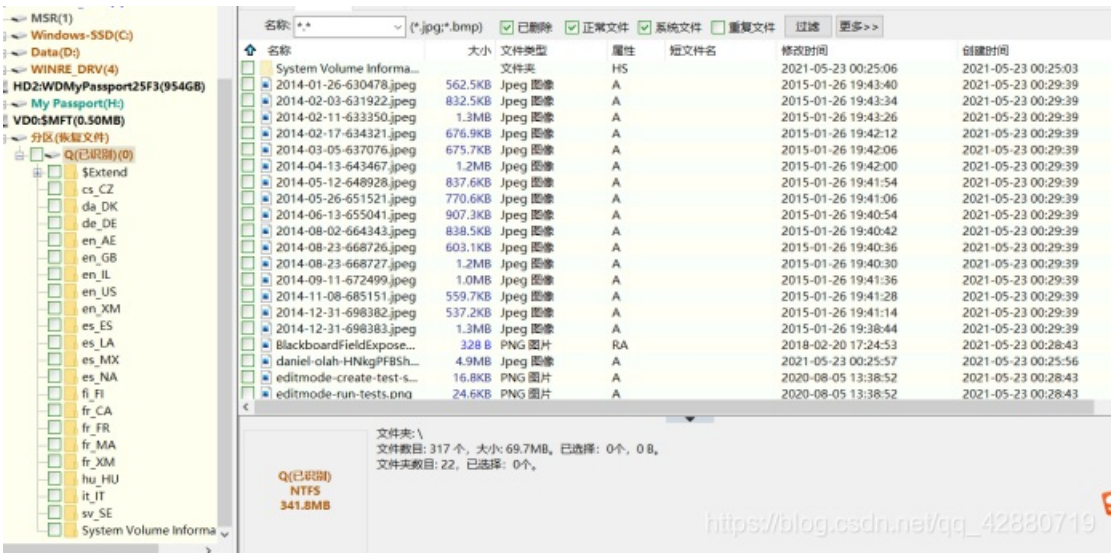
flag{P0w5rFu1_0pEn_50urcE}

EzTime

MFT能用diskgenius直接看，方法如下：







注意属性为A，都是不能导出的，但是这题问的是名字，所以已经够用了
$LogFile用github的LogFileParser能打开，可是已经不需要了
就这个MFT，一个个试不同时间的就行了。

{45EF6FFC-F0B6-4000-A7C0-8D1549355A8C}.png
为什么不是其他的，我也不知道，反正试错了好多个。